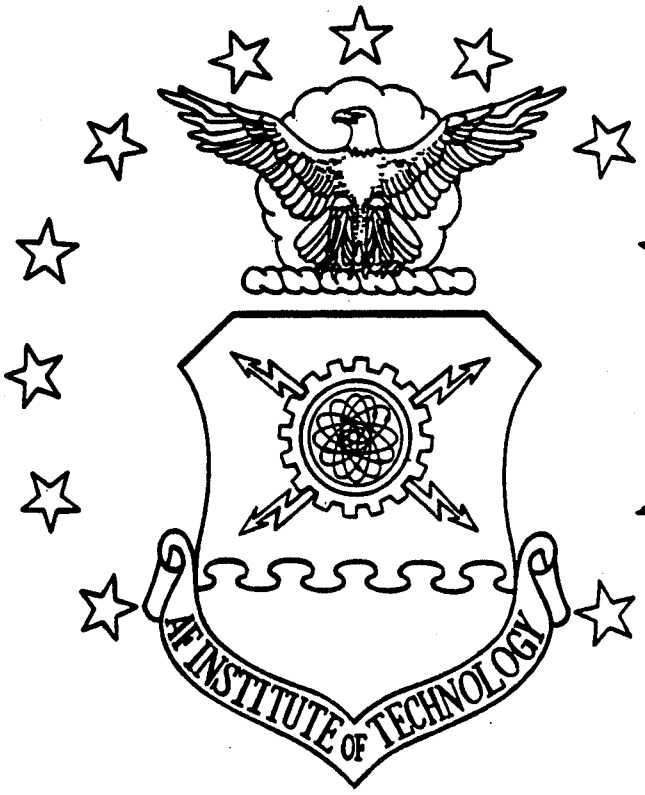


Embedology and Neural Estimation for Time Series Predicti

THESIS

Robert E. Garza
Second Lieutenant, USAF

AFIT/GE/ENG/94D-11

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/94D-11

Embedology and Neural Estimation for Time Series Prediction

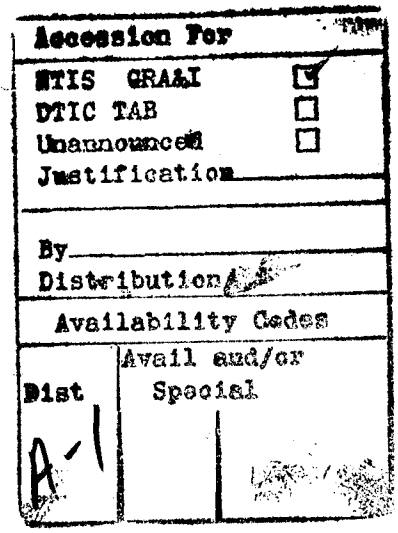
THESIS

Robert E. Garza
Second Lieutenant, USAF

AFIT/GE/ENG/94D-11

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.



AFIT/GE/ENG/94D-11

Embedology and Neural Estimation for Time Series Prediction

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Robert E. Garza, BSEE
Second Lieutenant, USAF

December, 1994

Acknowledgements

I could not have completed this thesis without the assistance of many people. Dr. Steven Rogers, my thesis advisor, showed me how to enjoy the work even while struggling to learn. Dr. Dennis Quinn provided support and several suggestions on how to continue my research. Dr. Dennis Ruck provided some invaluable suggestions at crucial points of my thesis. Dr. Joe Sacchini provided code to run the spectal estimation technique I needed, when it would have taken much too long to develop myself. My fellow students Rich Sumner, Lori Thorson, and Dennis Wolstenholme helped me keep my head above water when I really thought I was going to drown. I want to especially thank Georgia Harrup for her help with probability estimation and for always laughing at my jokes. Finally, I want to thank my family for all their support which has helped me to finally finish this thesis.

Robert E. Garza

Table of Contents

List of Figures

List of Tables

List of Symbols

Abstract

Time series prediction has widespread application, ranging from predicting the stock market to trying to predict future locations of scud missiles. Recent work by Sauer and Casdagli has developed into the embedology theorem, which sets forth the procedures for state space manipulation and reconstruction for time series prediction. This includes embedding the time series into a higher dimensional space in order to form an attractor, a structure defined by the embedded vectors. Embedology is combined with neural technologies in an effort to create a more accurate prediction algorithm. These algorithms consist of embedology, neural networks, Euclidean space nearest neighbors, and spectral estimation techniques in an effort to surpass the prediction accuracy of conventional methods. Local linear training methods are also examined through the use of the nearest neighbors as the training set for a neural network. Fusion methodologies are also examined in an attempt to combine several algorithms in order to increase prediction accuracy. The results of these experiments determine that the neural network algorithms have the best individual prediction accuracies, and both fusion methodologies can determine the best performance. The performance of the nearest neighbor trained neural network validates the applicability of the local linear training set.

Embedology and Neural Estimation for Time Series Prediction

I. Introduction

1.1 Historical Background

Recorded data gives a general idea of how a system is operating without actually explaining how the data is generated. The dynamical system which generates the data is only implied, not given. The time series, a discrete time-ordered set of real numbers corresponding to a single component of a dynamical system, does not define the underlying dynamics which will allow for accurate prediction of its behavior [30]. The time series can be used, however, in order to determine an approximation of the underlying dynamics of the dynamical system.

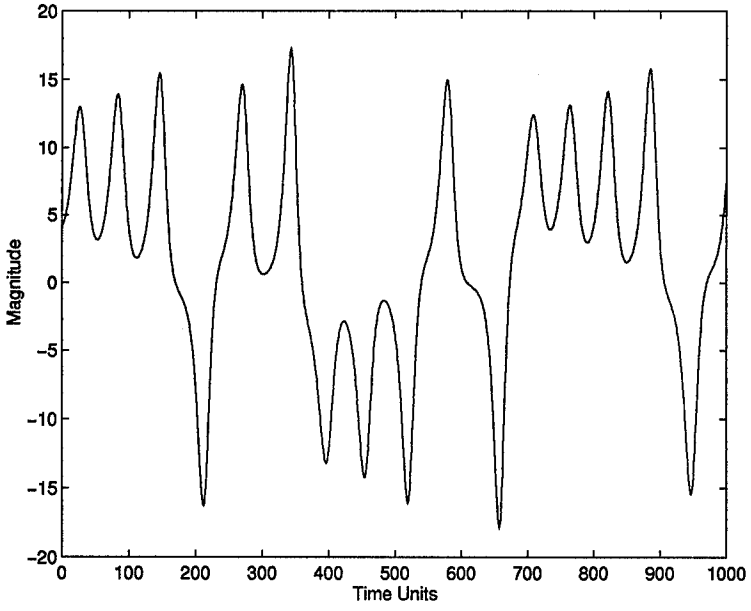


Figure 1. The Lorenz Time Series Approximation [27]

For example, the time series seen in Figure 1.1 is a 1-D approximation of the Lorenz attractor. The Lorenz system is made up of three differential equations.

$$\dot{x} = \sigma(y - x) \tag{1}$$

$$\dot{y} = \rho x - y - xz \tag{2}$$

$$\dot{z} = -\beta z + xy \tag{3}$$

with the parameters at the values set by Lorenz [20, 27].

$$\beta = 8/3, \rho = 28, \sigma = 10 \tag{4}$$

The solution to this system of equations reveals the sensitivity to initial conditions which characterizes a chaotic system.

A trajectory of the Lorenz attractor can be generated by using a differential equation solver, and a time series can be extracted by sampling a single coordinate. In this example, the x-coordinate is sampled with a period of $\Delta t = 0.05$ [27]. This time series, as seen in the figure, tells nothing of the differential equations which determine it, yet can be used to model the underlying dynamics of the generating system.

How can a time series be used to develop a prediction algorithm without knowing the series' generating principles? One method is through the use of embedology. In simple terms, embedology algorithms place a time series into a high dimensional state space, with respect to the space which is used to currently represent the data, so that they can be projected down onto locally generated regression hyperplanes. By determining the movement of the nearest points in state space, the movement of a particular point can be approximated, in order to forecast its position in the future [27].

As may be expected, some time series are predictable with a greater accuracy than others. There are many factors which determine the degree of predictability. A dynamical system with poorly understood characterizing factors is less deterministic than one with controlled factors [30]. A system with poorly understood influences is a chaotic time series [30]. Based on this definition, a chaotic time series is harder to predict than a more stable system.

As greater understanding of chaotic processes occurs, the inherent sensitivity to initial conditions of a chaotic dynamical system is easily seen. This sensitivity to initial conditions destroys long term predictability in the sense that, given an approximation to an object's phase space location, the divergence of nearby trajectories renders worthless any attempt to infer future nearness from current nearness [11, 30].

It is possible to accurately predict time chaotic time series, as shown by Martin Casdagli and Tim Sauer. Casdagli presented the first working algorithm, the Deterministic Versus Stochastic algorithm, in order to develop a more accurate nonlinear prediction method for use with chaotic aperiodic data [1]. A nonlinear, variable smoothing parameter exploits the inherent make up of the time series for forecasting purposes. The algorithm uses knowledge of the behavior of portions of the embedded data in order to form predictions. Prediction accuracy is related closely to the low dimensional chaotic nature (or lack of it) of the time series.

1.2 Problem Statement

Casdagli's Deterministic Versus Stochastic (DVS) algorithm will be improved to provide more robust prediction. Several improvement algorithms are developed: phase component confidence determination, Deterministic Versus Stochastic nearest neighbors as neural training set, Deterministic Versus Stochastic fusion with neural networks, and Bayesian Classification.

1.3 Scope

Casdagli's Deterministic Versus Stochastic algorithm has been refined by Dr. Tim Sauer, in a successful attempt to improve the prediction algorithm [1, 27]. This algorithm is based upon short term prediction of a time series. Both algorithms exploit the reconstruction of the state space of a dynamical system using delay coordinates, based upon work done by Ruelle and Takens [1, 24, 27, 32].

Attempts to improve Casdagli's algorithm stem from the need for more accurate prediction algorithms. The United States Air Force currently has several problems which require an accurate time series prediction method to provide an adequate solution. One such problem is SCUD identification, which employs a time series made up of multiple looks at projected targets. Another is anti-air capabilities, which need an accurate, time-critical determination of where the threat is going based upon its present and previous location.

The attempts to improve upon Casdagli by this thesis are restricted with respect to data set and method. Data sets are restricted to Standard and Poors financial data and the International Financial Forecasting Competition data [33]. The financial prediction area is one of the most difficult problems currently being researched. By restricting the data sets to financial data, the algorithms will have a measurable objective. In this way, the research done here for use in other areas will have an accuracy level comparable with other algorithms currently being researched.

The algorithms themselves are restricted to four areas. The first is the phase component confidence determination for the Casdagli algorithm. Limited success has been obtained through the application of Fourier series components as a measure of the accuracy of the Casdagli algorithm. The second algorithm is the use of the nearest neighbors, as determined by the Deterministic Versus Stochastic algorithm, to train a neural network. Pruning nearest neighbors using Casdagli's Deterministic Versus Stochastic algorithm did not significantly affect the prediction accuracy. The nearest neighbors should provide a basis for training, without having to use all of

the training data for neural adaptation. The third is to use the fractal dimension, found by the Deterministic Versus Stochastic algorithm, to determine the number of time components needed to successfully train a neural network. Captain Jim Stright had some success in classification using a similar process [30]. The conjecture is that it will also apply to neural training. The fourth is Bayes Bounding the data sets in order to determine the range of the error probabilities. This allows for comparison of specific algorithm prediction accuracies to the upper bound of the error probability range.

1.4 General Approach

Since the thesis data set is restricted to financial data, issues present in the financial engineering field become applicable to the current problem. One question is how underlying trends can be determined using a point-by-point prediction algorithm, such as Casdagli's Deterministic Versus Stochastic. Stock market analysts state the main goal of the investor as avoiding selling out of uptrends and buying into downtrends. Unfortunately, a point-by-point prediction algorithm, in itself, does not take this 'big picture' into account. This is an inherent problem in the point-by-point solution itself, because the question cannot be answered using any of the current algorithms. This problem is relevant for all prediction cases, since underlying trends need to be known to completely characterize any system.

A preliminary suggestion is to use the phase components of the time series to increase the confidence of the prediction. The phase components were determined through the use of Fourier series expansion. The current approach is to determine the phase components through the use of a parametric spectral estimator, in order to verify the Fourier expansion findings and reduce computation time.

Another problem is processing time. What good is a daily predictor that is ninety-nine percent accurate if it takes twenty-five hours to compute due to the computational complexity? To reduce the complexity, and therefore save time, the

nearest neighbors are examined to determine if they can be used as the neural network training set without losing any accuracy. This would substantially reduce the training time for the neural net.

One last problem is how to apply a nonlinear model to a time series. The simple solution to this problem is to use a neural network. The set-up of the neural network should resemble that of Captain Stright's classifiers used in his dissertation. The fractal dimension of the time series is used to determine the number of past samples that the neural network needs to train on. This should optimize the prediction accuracy for the time series if several neural net configurations are examined.

Bayes Bounding the data set allows for comparison of the bounds to the algorithms' prediction accuracies. Knowing the error probability range allows for the results of each algorithm to be compared to the error region. Each algorithm is analyzed according to the margin between its resulting prediction accuracy and the upper Bayes Bound for the data set. The comparison determines whether the algorithm can be improved, or whether it already minimizes the error probability for the data set.

1.5 Thesis Organization

Chapter I has provided a brief historical perspective of time series analysis, a statement of the research goal, and an outline of the approach to time series prediction. Chapter II will examine background material essential to understand the bases for the experiments presented in chapter III. Chapter IV will discuss the results of the time series prediction experiments. Chapter V will present conclusions and recommendations based upon the results of the research.

II. Background Material

2.1 Introduction

Chapter I provides a brief historical view of the time series prediction problem and the goals of this research. This chapter will provide the background material necessary to understand the prediction algorithms developed in this thesis. This includes an in-depth examination of the following topics: (1) the fractal dimension of the time series, a measurement developed to characterize the inherent nature of the time series, (2) the chaos predictor developed by Casdagli, an algorithm which determines the inherent nature of the time series to provide better prediction capabilities, (3) parametric spectral estimation, a process which determines the phase of a time series, (4) nearest neighbors and state space, the determination of the embedding space of the time series using the points closest to the hyperplane, (5) artificial neural networks, a processing tool which allows for nonlinear approximation. These tools allow prediction strategies to be combined in order to allow for classification of the movement of the prediction point. Knowing where the prediction point is moving is essential to developing an actual number prediction algorithm.

2.2 Fractal Dimension

The fractal dimension is a measurement which allows for the characterization of a dynamical system in terms of its chaotic nature. Chaotic systems are those which are not (multiply) periodic and are unpredictable over long times, being extremely sensitive to initial conditions [12]. One example of a chaotic system is a degenerative laser, whose response is seen in Figure 2.1. The laser response can be described as a function f and a countably infinite number of derivatives f', f" , ... The set of all function values constitutes the phase space approximation of the system.

Phase space is the dimensional space which completely contains an attractor. For example, a time series may be defined as the functional approximation for a

given time period τ [20].

$$t_0 = t(0), t_1 = t(\tau), t_2 = t(2\tau), \ldots \tag{5}$$

Now choose a time delay T, a multiple of τ, such that a series of vectors of dimension m is defined [20].

$$(t(0), t(T), t(2T), \ldots, t(MT)) \tag{6}$$

$$(t(\tau), t(\tau + T), t(\tau + 2T), \ldots, t(\tau + MT)) \tag{7}$$

$$\vdots \tag{8}$$

$$(t(p\tau), t(p\tau + T), t(p\tau + 2T), \ldots, t(p\tau + MT)) \tag{9}$$

Plotting these (p+1) points in (m+1) dimensional space with connecting line segments, the essential features of the attractor are apparent [20]. The method of choosing the dimension size m is discussed in Section 2.3.

The laser's response thus approaches an attractor, a subset of the phase space. In a system with F degrees of freedom, an attractor is a subset of F-dimensional phase space towards which almost all sufficiently close trajectories get 'attracted' asymptotically [12]. Strange attractors are attractors which characterize the final state of dissipative systems which are highly complex and exhibit all the signs of chaos [20]. Dissipative dynamical systems, such as the pendulum, are those which lose energy over time.

In efforts to define the nature of strange attractors, the Lyapunov exponents have been extensively studied. The Lyapunov exponents, λ_i, are determined by

$$\epsilon_i(t) \approx \epsilon_i(0) \exp \lambda_i t \tag{10}$$

where $\epsilon_i(t)$ defines the principle axes of an ellipsoid constructed of an infinitesimally small F-dimensional ball in phase space [12]. The sum of the λ_i's have to be negative

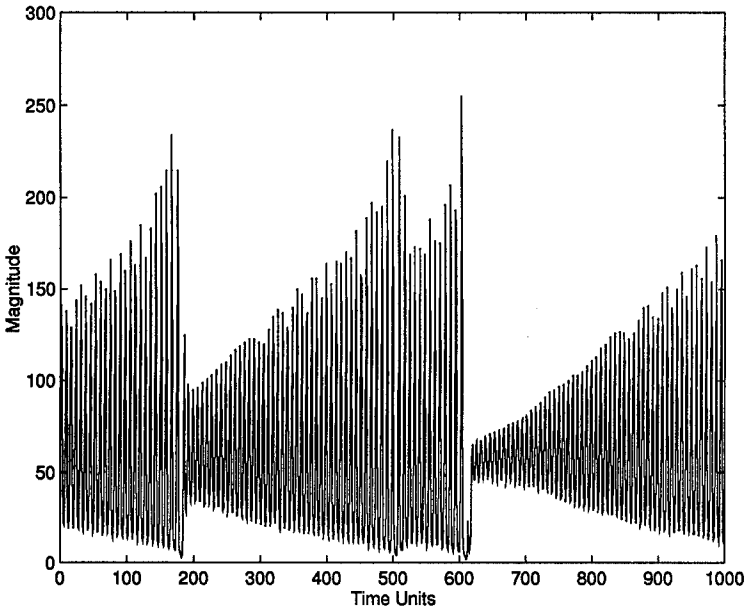


Figure 2. Laser Data obtained from the Santa Fe Competition [27]

since they describe the contraction of volume [12]. However, the strange attractor results from the stretching and folding of the volume into a fractal structure, which guarantees that at least one of the λ_i's is positive. The drawbacks are that the Lyapunov exponents are not easily measured and that they do not describe the volumetric folding action used in the generation of the strange attractor.

The fractal dimension is thus a measure of the local structure of strange attractors. To define the fractal dimension, first cover the attractor with F-dimensional hypercubes with sides of length l and consider the limit as $l \rightarrow 0$. If the minimum number of cubes needed for the covering grows like

$$M(l) \simeq l^{-D} \tag{11}$$

then the exponent D is the fractal dimension [12]. Solving for D gives

$$D = \lim_{l \rightarrow 0} \frac{logM(l)}{|log(l)|} \tag{12}$$

Chaotic attractors typically have fractal dimensions in the range of 1.95 to 7.5 [30].

Grassberger and Procraccia provide an algorithm which embeds time series data into phase space and finds the fractal dimension of the data [12]. Casdagli's Deterministic Versus Stochastic algorithm may also be used to determine the fractal dimension using the scaling law defined as:

$$E_m(k) \approx C(\frac{k}{N_f})^{\frac{2}{D}} \tag{13}$$

where E_m is the normalized root-mean square (RMS) forecasting error, C is a 'curvature' constant, N_f is the number of elements in the fitting set of the data, and D is the fractal dimension [1].

The fractal dimension is necessary in that it plays a critical role in embedding the time series into the correct dimension of the state (phase) space to allow for complete reconstruction. These techniques of state space reconstruction, first introduced by Packard *et al* and Takens, show that it is possible to address the problem of dimension estimation by direct observations of sufficiently long time series of the systems of interest [1, 19, 32]. Takens proved that if the embedding dimension, m, is greater than twice the fractal dimension, then the delay vectors of the embedded time series fill out a reconstructed attractor in $\Re^m$ which is diffeomorphic to the original attractor. Further work by Sauer *et al* show that, for

$$D < m < 2D \tag{14}$$

a reconstructed attractor can be obtained almost anywhere [1, 26].

2.3 Chaos Predictor

Nonlinear modeling and forecasting of time series data is a relatively new field, with stochastic nonlinear modeling introduced about 1980 and deterministic nonlinear models introduced about 1987 [1]. Casdagli's Deterministic Versus Stochastic

algorithm is the first attempt to bridge the gap between the stochastic and deterministic approaches. Casdagli surmises that if models nearer the deterministic end of the algorithm give the most accurate short-term forecasts, then the data contains low dimensional chaotic behavior [1].

Casdagli focused his efforts on forecasting algorithms, which incorporate state space reconstruction and fractal dimension calculations. The main focus of the research, to determine whether a high dimension chaotic system is equivalent to a stochastic system, will determine if a single algorithm can thus distinguish between low dimensional chaotic dynamics and stochastic dynamics [1]. This premise is based upon several experimental findings.

If the effects of noise are small, and the fractal dimension is relatively small, then a modest amount of data, approximately 10^D, can find an accurate approximation for f. This is tested using out-of-sample short term forecasting, obtaining more accurate results as tested than if a substantial stochastic component is present [1]. Conversely, if the fractal dimension is relatively large, the same amount of data will not be able to approximate the equation with a deterministic model [1]. This supports the premise that a high dimensional system is equivalent to a stochastic system.

A stronger argument in favor of Casdagli's premise is that high dimensional chaotic systems with low noise can induce a large noise term when using Taken's Rule for state space reconstruction [1]. Previous knowledge of the scaling parameters and the smoothing function do not help to improve the accuracy of short term forecasting no matter the length of the time series. This argument in favor of Casdagli's premise substantially strengthens the argument that low dimensional chaos can be distinguished from stochastic dynamics.

Casdagli's algorithm, based upon the Farmer-Sidorowich algorithm, is fairly simple [7]. The time series is divided into two sets, N_f, the fitting set, and N_t, the testing set. Find the distances d_{ij} between the test vector, x_i, and the delay vectors

made up of the fitting set. Order d_{ij} from smallest to largest. Find the k nearest neighbors, and fit the linear model

$$x_{j(l)+T} \approx \alpha_0 + \sum_{n=1}^{m} \alpha_n x_{j(l)-(n-1)\tau} \tag{15}$$

for $l = 1, \ldots k$ to calculate $x_{j(l)+T}$. This is repeated for several k values in order to determine the correct number of nearest neighbors. Use the model to estimate a T-step-ahead forecast $\hat{x}_{i+T}(k)$ for the test vector x_i. Compute the local error $e_i(k)$. This is repeated to forecast in the entire testing set and compute the root-mean-square (RMS) forecasting error [1].

For example, consider the Standard and Poors data set. It contains the closing values for the Standard and Poors index, an average value of a number of stocks used to show the trends of the stock market. The input requirement for the Casdagli algorithm is a single vector of data point, such as the data seen here.

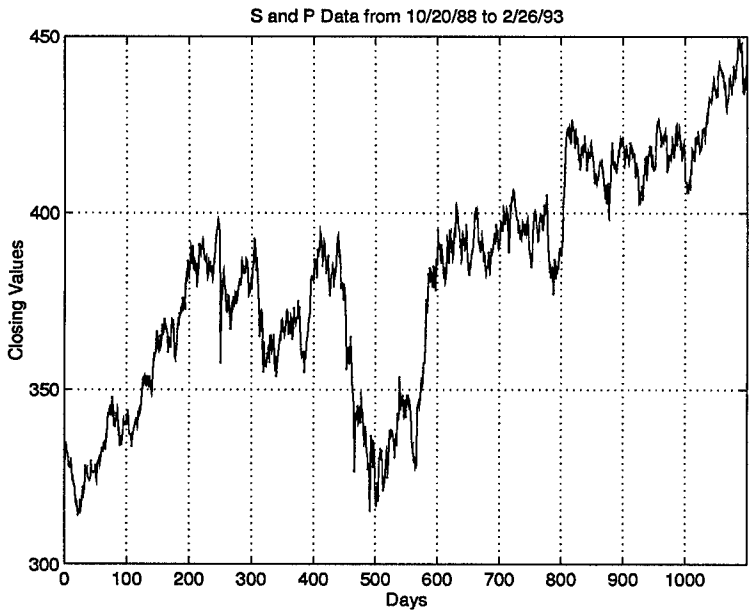


Figure 3. The Standard and Poors Data

The Casdagli algorithm queries the user for the embedding dimension, m, the number of data points in the fitting set, N_f, and the number in the testing set, N_t. It also requires the τ and the T to perform the prediction. The first step the algorithm takes is to determine the nearest neighbors, in Euclidean space, for the final point of N_f. The number of nearest neighbors, k, is varied in order to determine the optimal k. This value can then be fixed to optimize the prediction performance.

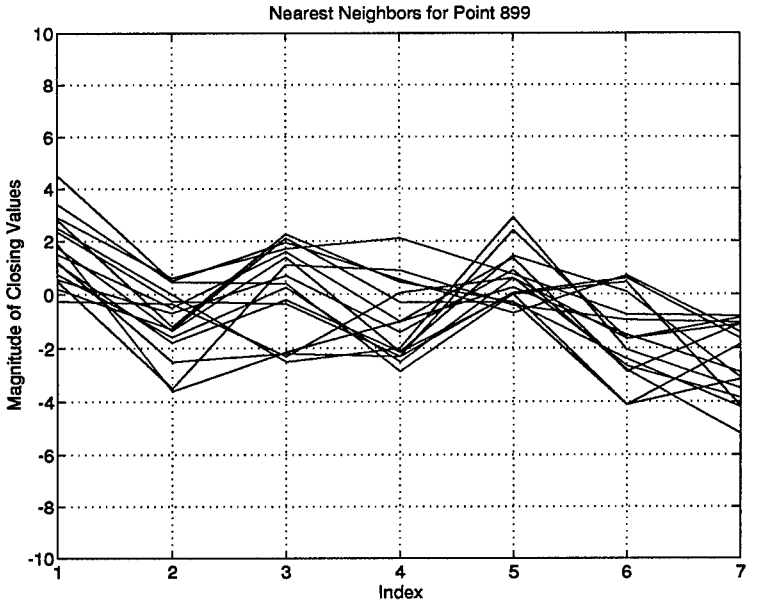


Figure 4. The Nearest Neighbors for k=16 for the Standard and Poors Data

The Casdagli algorithm uses least squares estimation to predict the next value of the time series. This algorithm places a linear regression hyperplane through the nearest neighbors. This allows for the prediction point to be estimated on the hyperplane, based upon on the movements of the neighbor's trajectories toward the hyperplane.

This process is repeated for each prediction point. Remember that the number of nearest neighbors, k, is now fixed. This is based upon studies done by Stright which conclude that the k found for a local area is representative of the k for the

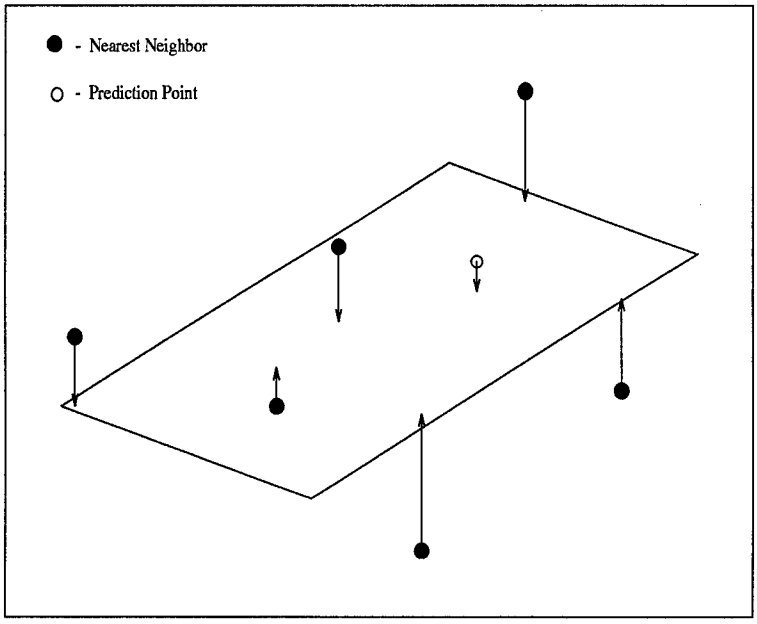


Figure 5. The Prediction using a Linear Regression Hyperplane

whole series [31]. This allows for one k to be used for prediction no matter what part of the series is being analyzed.

2.4 Parametric and Spectral Estimation

In determining the spectral components of a time series, there are two approaches which can be taken. The first is using a classical spectral estimation technique, such as the periodogram or the Blackman-Tukey spectral estimator. The second is the parametric spectral estimation techniques, such as the autocorrelation or modified covariance method [15].

The classical approaches for spectral estimation use Fourier transform operations on either windowed data or windowed autocorrelation function (ACF) estimates [15]. Windowing data in such a fashion smears the spectral estimate, due to the assumption that the data, or autocorrelation function, values outside of the window are zero. This smearing causes the classical techniques to become unreliable, los-

ing most of the spectral components to noise even when the window is large. The parametric techniques prove to be more reliable.

Use of *a priori* information may permit the selection of an exact model for the process which generated the data samples, or at least a model that is a good approximation of the actual underlying process [15]. This allows for a better spectral estimation based upon the model. This approach allows for the assumptions concerning the data to change, eliminating the need for the data outside of the window to be zero or cyclic. This elimination of windowing brings parametric techniques, such as autoregressive (AR) modeling and the Prony Method, to the forefront [15].

Autoregressive (AR) spectral estimators exhibit poor performance when applied to sinusoids in noise. The Prony Method also performs poorly in the presence of noise, although superior to other methods in the same situation. If the noise is not white Gaussian, the least squares estimate does not produce an approximate maximum likelihood estimate of the amplitude coefficients, thereby rendering the model ineffective [15]. The Prony Method will perform well if there is no noise, or if the data can be modelled as a signal mixed with white noise

$$d[n] = f[n] + w[n] \tag{16}$$

where d[n] is the modelled data, f[n] is the signal, and w[n] is white, Gaussian noise [25]. The Prony Method works in these cases because the maximum likelihood estimate is well approximated by the least squares method, using the Singular Value Decomposition (SVD).

The Least Squares (TLS) Prony's Model is given by equation 17.

$$d(n) = \sum_{k=1}^{K} a_k p_k^n, n = 0, 1, \ldots, N-1 \tag{17}$$

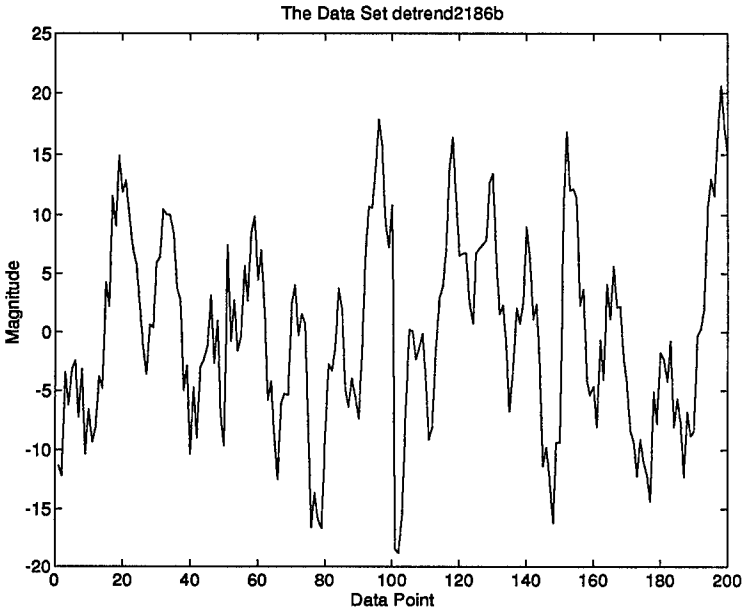


Figure 6. A Sample Signal Usable with the TLS Prony Method

where p_k is the kth pole, a_k is the kth amplitude coefficient, and K is the number of modes [25]. Prony's Model models the frequency domain data, to estimate the poles and amplitude coefficients from the given data. A backward linear prediction approach, utilizing a Singular Value Decomposition (SVD) based noise cleaning algorithm, determines the pole estimates. An in-depth algorithm is presented by Sacchini in his dissertation [25]. Several examples presented by Sacchini demonstrate the utility of parametric techniques, especially the TLS Prony Model, to determine the spectral components of a time series [25].

Running the TLS Prony Method, as defined by Sacchini [25], results in a series of numbers which define the poles and amplitude coefficients of the major phase components of the data. The data is presented as a single vector of the time series values. A sample output of the TLS Prony Method shows the amplitude coefficients and the poles of the determined phases. Remember that poles occur in pairs, as defined by the Fourier Transform of sinusoids.

Phase of Pole	Magnitude of Pole	Amplitude Coefficient
-0.0300	1.0004	2.5360
0.0300	1.0004	2.5360
-0.0824	0.9950	3.7776
0.0824	0.9950	3.7776
-0.3509	1.0317	0.0038
0.3509	1.0317	0.0038

Table 1. Sample Output from the TLS Prony Method

To remove spurious poles, look at the magnitude of the pole and the size of the amplitude coefficient. A pole magnitude of 1.000, on the unit circle, is the best approximation available. A pole which is not on, or really close to, the unit circle is usually an error due to an incorrect model order. Looking at the example above, the pole located at 0.3509 does not really exist. Now consider the amplitude coefficient. The amplitude coefficient of a pole should be as large as possible. If an amplitude coefficient is too small, determined on a case-by-case basis, then the pole should be removed. Notice in the example that the pole removed for its pole magnitude will also be removed for its small amplitude coefficient.

The poles determined by the TLS Prony Method can be used in an attempt to reconstruct a given time sequence. The poles are the component sine waves of the reconstructed signal. If the TLS Prony Method accurately determines the phase components of the time series, it can give an accurate approximation of the original signal. If the phase determination is done using a noisy signal, the accuracy of the reconstruction will be reduced, since the sine wave components of the signal will be distorted in their determination.

2.5 Nearest Neighbors in State Space

The use of the nearest neighbors in state space is the foundation of the the embedology approach to prediction. The present state of the system is identified, and similar previous states are sought. The evolution of the time series of the

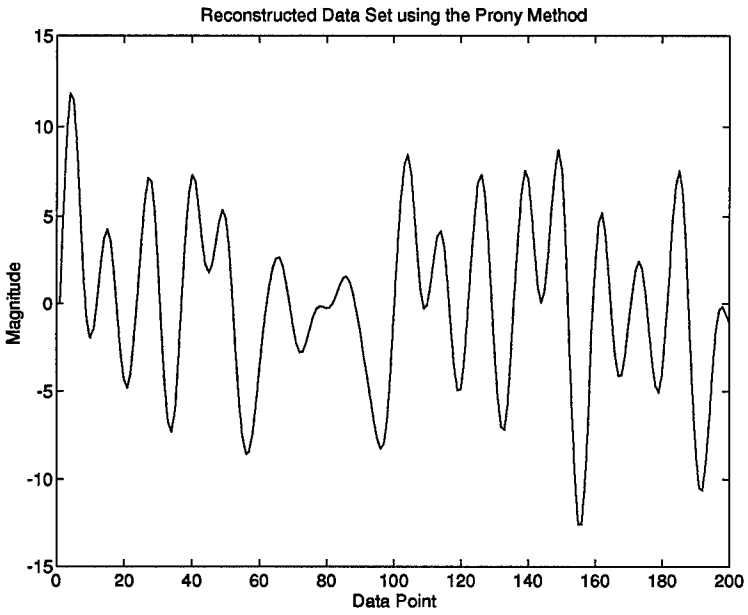


Figure 7. A TLS Prony Method Reconstruction of the Sample Signal

previous states yields information concerning the future [27]. The nearest neighbors are relatively straightforward to calculate. The time series is embedded into the space $\Re^m$, which is determined through the use of the fractal dimension and Taken's Rule [1, 32]. The present state $\bar{x}$ is the vector of length n which constitutes the present point in $\Re^m$. To do the search for the k nearest neighbors, the past samples of the time series are broken up into m-tuples, and compared to $\bar{x}$ using the Euclidean distance measure.

$$d(\bar{x}, \bar{y}) = \sqrt{(\bar{x} - \bar{y})^T(\bar{x} - \bar{y})} = +\sqrt{\sum_{i=1}^{m}(x_i - y_i)^2} \tag{18}$$

The k m-tuples with the smallest values for d are chosen as the nearest neighbors. Thus, for a time series with n=7, the data is broken up into its constituent 7-tuples. These vectors are then compared to the present vector $\bar{x}$. The k closest vectors are chosen, giving a data set of size k x 7.

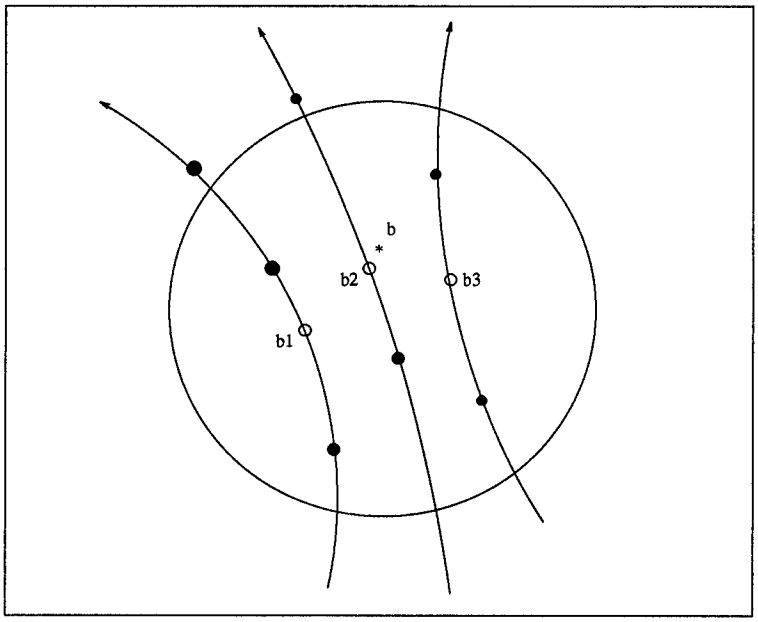


Figure 8. The Nearest Neighbors to b in reconstructed State Space [27]

The main issue concerning the nearest neighbors is the determination of k, such that the error is minimized using various algorithms. The easiest way to determine the proper value for k, as seen in Casdagli's Deterministic Versus Stochastic algorithm, is to run the algorithm several times for the same point using varying values of k [1]. The error can then be examined to determine the best value for k, which would be the value which gives the smallest error. Using this algorithm will lead to the generation of a list of nearest neighbors versus error. In the sample search, k would be set equal to 16. This k can then be used as the k value for local prediction,

number of nearest neighbors, k	Mean Squared Error
8	0.3980
16	0.2805
32	0.3573
64	0.3769
128	0.3904
256	0.4030

Table 2. Sample Output using the Search Method for the Determination of k

since an optimal k value often changes after a short window of time [31].

This leads to the question of whether a local k could be further optimized for each particular prediction run. The answer to this question, as shown by Stright in his dissertation, is yes and no. Yes, each particular prediction run can prune the number of nearest neighbors, starting at k, to optimize its performance. No, in that this methodology does not significantly improve the prediction error [31].

2.6 Artificial Neural Networks

The neural network is used to fit a nonlinear surface to the problem of time series analysis, as seen in Figure 9. Neural networks are able to generalize the function of a time series through adaptation of the times series data. Once the neural network generalizes the function, new points can be presented to the network for classification. The accuracy of the neural network is based upon how well it can determine the underlying function of the time series.

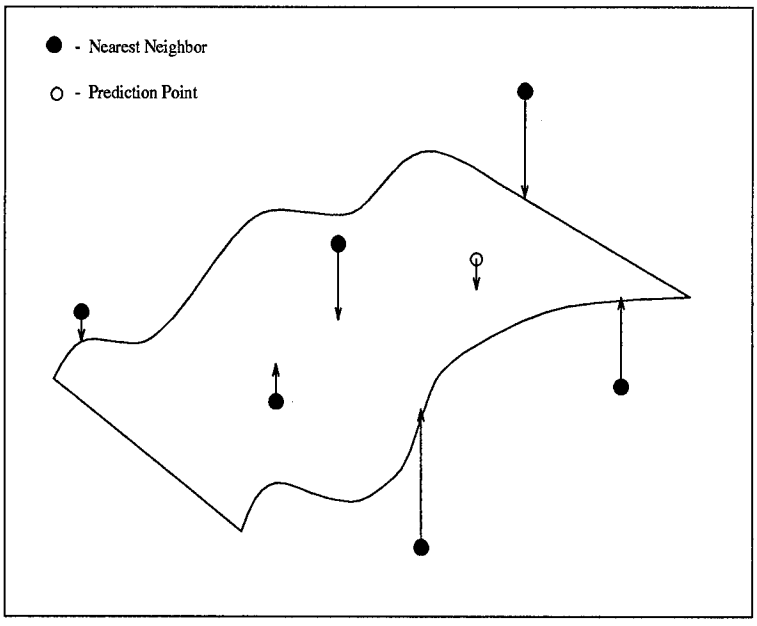


Figure 9. Fitting a Nonlinear Surface Through the Nearest Neighbors

Artificial neural networks are models composed of nonlinear processes operating in patterns reminiscent of biological neuron interconnections [17, 30]. The structure consists of an input layer, hidden layers, and an output layer. Cybenko and others have shown that one hidden layer is sufficient for any arbitrary transformation, given enough nodes, but several hidden layers may speed up processing time by decreasing learning time [6, 22].

After study of biological examples, researchers developed the perceptron, as in Figure 10, as an artificial neural approximation. The perceptron has N inputs, which are each multiplied by a weight in order to find the output value y. The weights are varied according to the importance of the input node to the output, since y is a function of the sum of the weights and a bias. Finding the weights which best approximate y is called "learning."

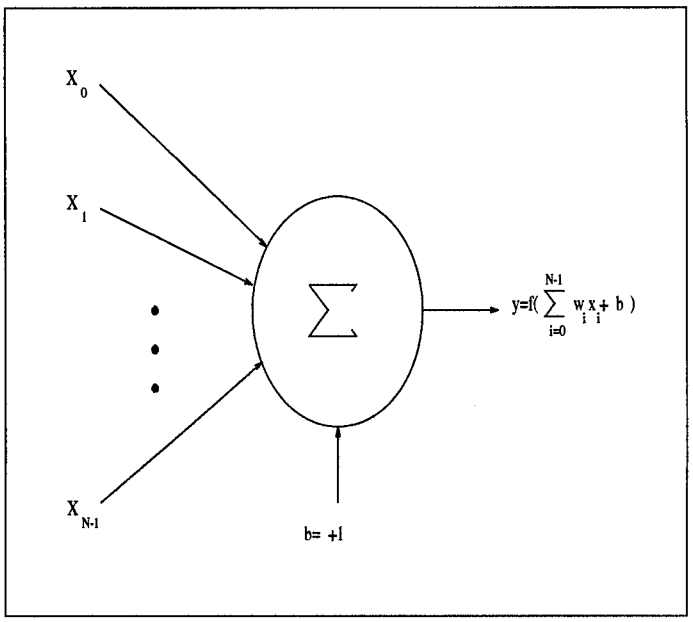


Figure 10. A Single Perceptron [30]

The learning procedure for a perceptron is fairly simple, using gradient descent to update the weights. First break the data up into three sets, the training set, the test set, and the evaluation set. Having randomly set the weights, the perceptron

receives a training vector and the desired output. The training vector's real output is calculated, and the weight update is computed using equation(2.15) [22].

$$w_i^+ = w_i^- + \eta(d - y)x_i \tag{19}$$

where w_i^+ is the new weight, w_i^- is the old weight, η is the step size, d is the desired output, y is the true output, and x_i is the training vector. This process is repeated for every training vector. This type of updating is known as instantaneous, since it occurs after each training epoch. After a specified number of training epochs, supply the weights to the test set to find the network's current capabilities.

Training occurs until the training error continues dropping but the test error starts to rise. To continue to learn would simply memorize the training data without generalizing for the function being sought. The weights are then fixed, and the evaluation set tested to find out how well the network approximated the data's underlying function [22].

This is the procedure for a single perceptron, which is fairly limited in size and scope. Thus evolved the multi-layer perceptron (MLP) architecture. The multilayer perceptron works on the same principles as the perceptron, except that there is an entire structure of neurons used to approximate the output function.

The multilayer perceptron (MLP) is a static network which passes a weighted sum of the inputs through a nonlinearity, in this thesis a standard sigmoid function.

$$f_{sigmoid}(y) = (1 + e^{-\beta y})^{-1} \tag{20}$$

y is the output of a hidden node and β is the gain of the sigmoid, equal to 1 in this thesis. The sigmoid's differentiability allows for the use of gradient descent learning algorithms [14]. The nonlinear sigmoidal function is also used in order to provide a better structure from which to derive the output function's approximation. One

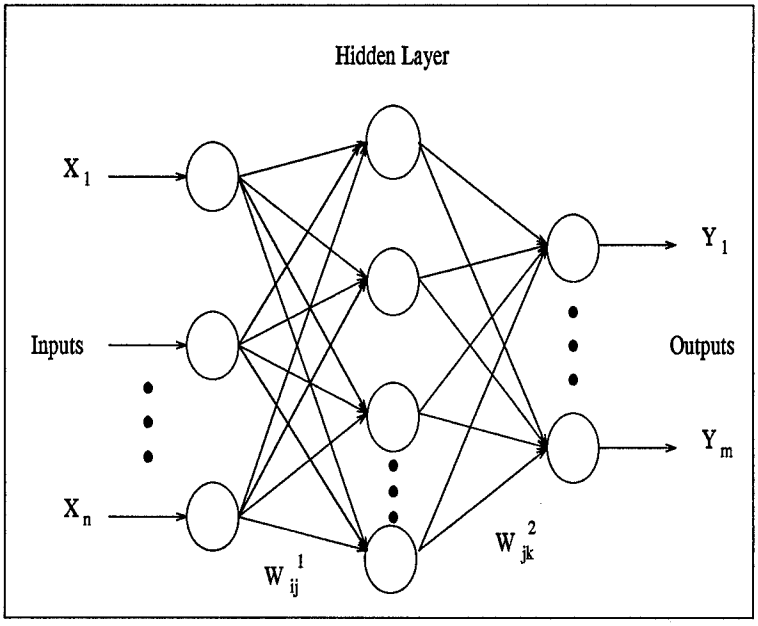


Figure 11. A Multi-Layer Perceptron Architecture [31]

big advantage is that the sigmoid is infinite, and will therefore form some type of approximation for areas in which no training samples occur [22].

Learning for the multilayer perceptron is based upon gradient descent, as is that of the single perceptron. The learning process occurs for every weight of every node of the multilayer perceptron, with different update formulas for the hidden and output layers. The formula for the hidden layer update is found in equation 21 and that for the output layer update is found in equation 22. These formulas are based on the architecture found in Figure 2.10 and the output nodes' use of nonlinear sigmoid functions for adaptation.

$$w_{ij}^{+1} = w_{ij}^{-1} - \eta \sum_{k=1}^{m} w_{jk}^{2}(d_k - yk) f_j(\bar{x})(1 - f_j(\bar{x}))x_i \tag{21}$$

$$w_{jk}^{+2} = w_{jk}^{-2} + \eta y_k(1 - y_k) f_j(\bar{x}) \tag{22}$$

Another class of neural networks is the Radial Basis Function (RBF) classifiers [22]. This network calculates discriminant functions using local Gaussian functions

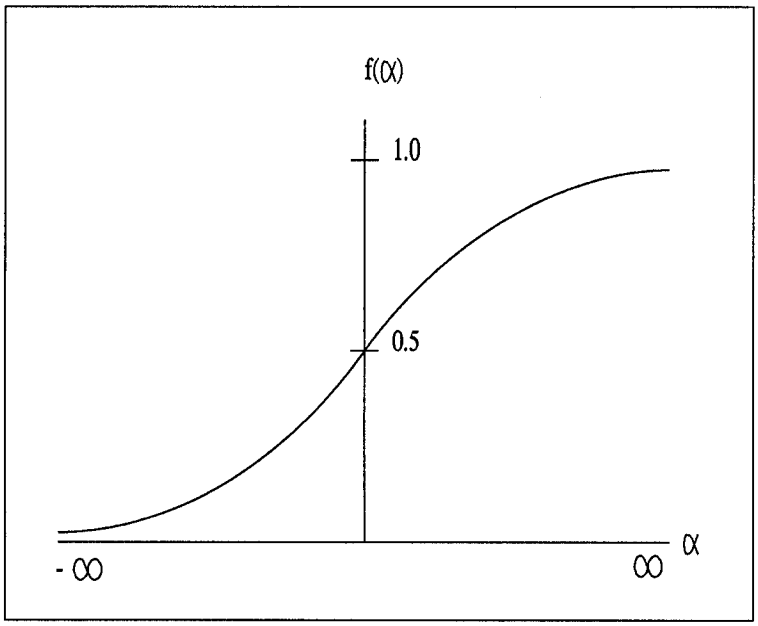


Figure 12. The Sigmoid Nonlinear Function

instead of sigmoids for the hidden layer. This network forms a linear combination of the kernel functions computed by the hidden layer nodes, which produces a localized response to input stimulus [14]. The kernel function of the hidden layer nodes, μ_{1j} is Gaussian.

$$\mu_{1j} = \exp[-\frac{(x-\omega_{1j})^T(x-\omega_{1j})}{2\sigma_j^2}] \tag{23}$$

μ_{ij} is the output of the jth hidden node, ω_{1j} is the weight vector associated with the jth hidden node, x is the input vector, σ^2 is the normalization vector for the jth hidden node, and N is the number of hidden nodes (j ranges from 1 to N) [14].

This has led to the establishment of a single hidden layer network, with the nodes in the hidden layer using radial basis functions to transform their inputs into their outputs. The most appealing characteristic of the RBF networks is almost instantaneous training times involved with setting the network parameters [34]. One problem with this classifier is that the function exists only where the training samples have occurred. If the test vector occurs in a previously unseen area, the output is no

better than a guess. The multilayer perceptron performs better in this case, since the approximation is based upon the infinite sigmoidal function [22].

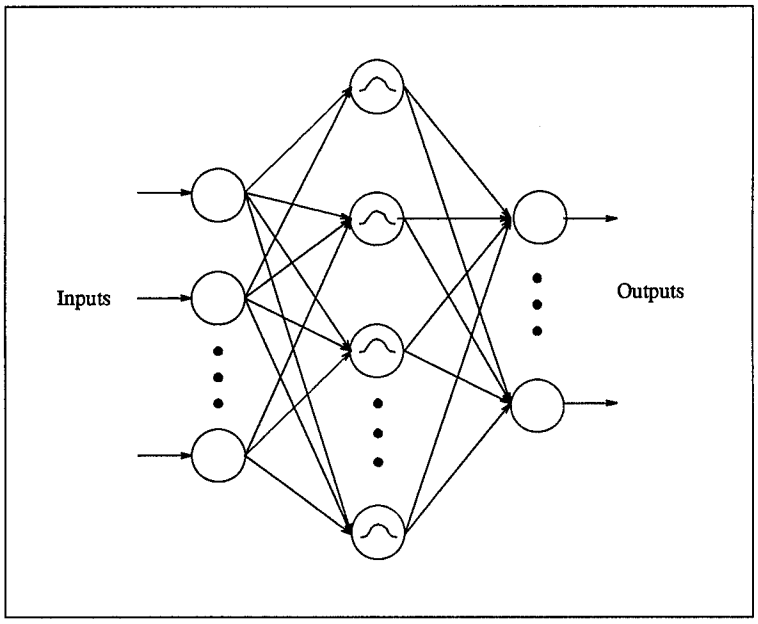


Figure 13. A Radial Basis Function Neural Network Architecture [14]

Several training rules apply to the use of neural networks. Two rules apply if the operator is prepared to accept a testing error of approximately ten percent [22]. Foley's rule states that if the number of training samples per class is greater than three times the number of features and the function is a multivariate Gaussian, then the density functions can be estimated [22]. Bernie's Rule states that the network will not memorize the training data if the number of training samples is greater than ten times the number of weights [22].

For the multilayer perceptron, Foley's rule states that ten times the number of weights should be less than the total number of training vectors. If the number of input nodes and output nodes is determined using the problems parameters, the number of hidden nodes can then be determined [22].

$$\frac{Tr}{10} \leq (In + Bn) * Hn + On * Hn \tag{24}$$

Tr is the number of training vectors, In is the number of input nodes, Bn is the bias node, On is the number of output nodes, and Hn is the number of hidden nodes. The bias node is a threshold value often set to 1. The number of training vectors per class is easily met with all data sets considered in this thesis. This setup will not memorize the training data with a minimum expected testing error of ten percent [22].

For the Radial Basis Function, the number of hidden nodes is determined by a clustering algorithm. An exhaustive search over K, the number of clusters, will determine the optimal number of clusters to reduce the training error rate. Once chosen, K is used for all networks using the same data set. The K-means clustering algorithm is used to cluster the data.

The K-means clustering algorithm is straightforward. Initialize the cluster centers ω_j, j=1,2,...,N1, with the clusters initially set to the first N1 training samples. All training samples are assigned to the closest clustering center using Euclidean distance as the metric. Compute the means of the clusters, and then move the cluster centers to the means. Recompute the assignment of the training vectors, find the means of the new clusters, and again move the cluster centers to the new means. Repeat this process until there is no change in the cluster assignments from one iteration to the next [14].

The architectures of the neural networks need to be set up correctly so that they will work properly. If too few nodes are used, the network is overwhelmed with information and never learns. Too many nodes allows the network to memorize the training data, which does not generalize the data's underlying function very well. In order to find a size more suitable to the problem, start at the maximum size and reduce the size of the structure. Find the structure that gives the minimum error for the minimum amount of training time, and use that structure for the remainder of the research.

Features are the inputs to the network which allow it to determine the non-linear boundaries which separate the classes. Schalkoff says that features are any extractable measure used [28]. This is true for time series prediction features, which range from economic indicators to the latest trends in the data. In this thesis, the features are kept the same for all of the neural networks used, including those in the nearest neighbor algorithms found in Section 3.5. The features are an m-tuple of the data, with m determined through the use of Casdagli's Deterministic Versus Stochastics algorithm (See Sections 2.3 and 3.3).

Now that the features and the architectures are determined, the Radial Basis Function networks are ready for use. The multilayer perceptron still has some parameters which need to be determined. The number of training epochs, the step size, and the momentum need to be assigned. First consider the step size. Properly setting the step size and momentum will properly apply the gradient descent training algorithm [4]. The step size, η, as seen in Equation 21, is a function of the number of training samples used. A small training set usually requires a larger step size while a large training set requires a smaller one. With a common step size of 0.1, a small training set might use a step size of 0.2 while a large training set might use a step size of 0.01.

The momentum, α, must be chosen to aid in learning. If the network is taking too long to learn, with a flat training error curve, the momentum needs to be increased [4]. This increases the size of each movement during the gradient descent, thus increasing the training rate. If the network is oscillating, or dropping down in a ragged manner, the momentum needs to be reduced [4]. This will keep the gradient descent algorithm from bouncing from one extreme to another, as it does when the momentum is too large.

Consider how long the neural network needs to be trained. One training epoch is a single presentation of the entire training set to the network, which sets the networks weights. As expected, training times differ greatly dependent upon the

data sets being used. Thus training times are discretionary. One criterion commonly used is to stop training once the testing set error starts to rise while the training error is still falling [22]. This fixes the weights when the network determines the smallest error.

With these parameters determined, the multilayer perceptron model is ready for use. Several network sizes are run, in order to determine which combination of parameters and network sizes determines the lowest test set error. The multilayer perceptron starts with the maximum number of hidden nodes and work down, in order to determine the best architecture to use for the particular data set and parameters. The Radial Basis Function networks will run the optimal K-clustering algorithm as determined by the exhaustive K-means search.

2.7 Probability Theory and Non-Parametric Estimation of the Bayes Error

Probability theory is a mathematical method used to describe random phenomena that can be approximately described by the relative frequency of the occurrence of the possible outcomes [13]. A simple example of this is tossing a fair coin. Although the outcome on any given toss cannot be predicted, it is likely that after many tosses, approximately half of the outcomes will be heads and the other half tails.

The probability density function (PDF) is a representation of the distribution of the outcomes of a random experiment. It shows the probability for observing any outcome on a given trial of the experiment. The PDF has the following properties [13]:

$$f(x) \geq 0 \quad \text{for all } x \tag{25}$$

$$\int_{-\infty}^{\infty} f(x)dx = 1. \tag{26}$$

The PDF for the coin toss mentioned above is discrete, based upon a summation rather than an integral, as seen in Figure 14.

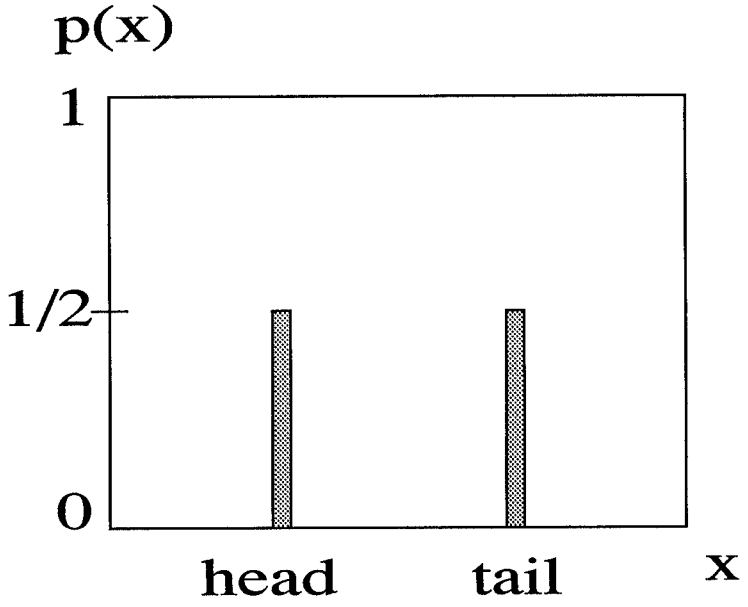


Figure 14. The Probability Density Function (PDF) for a fair coin toss [13]

Continuous PDFs are characterized by the probability of any given point occurring being 0. The probability of the outcome of the experiment falling between points a and b is the shaded area described by

$$\int_a^b f(x)dx. \tag{27}$$

An example of a continuous PDF is seen in Figure 15.

Once the PDFs are estimated, error probabilities can be found. When two PDFs are plotted against each other, a decision threshold (t) can be placed at any point on the axis based upon a particular decision rule. For example, given the PDFs in Figure 16, the decision rule is to choose S_1 if the outcome falls to the left of the decision threshold and S_2 if it falls to the right. Using the decision rule, the probability of error for S_1, any S_1 point that is classified as belonging to S_2, is the tail area of the S_1 PDF which lies to the right of the threshold. Similarly, the probability

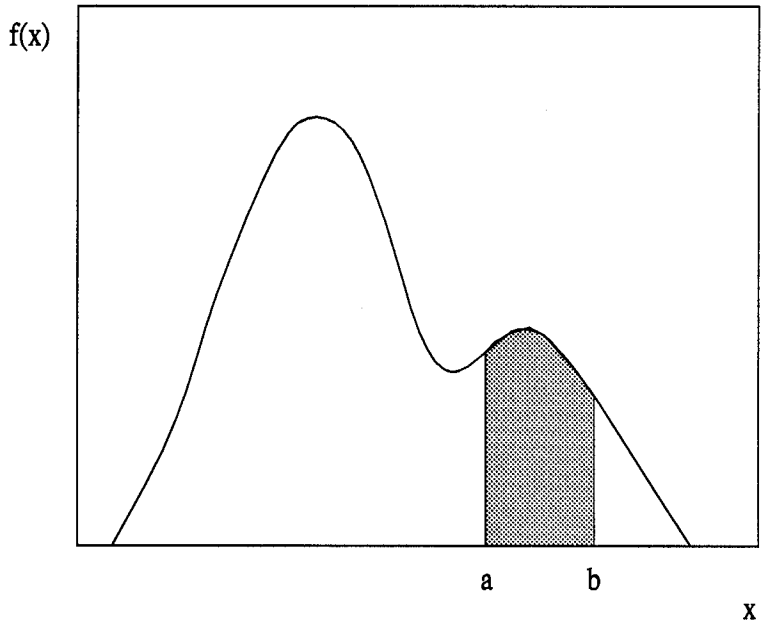


Figure 15. A Continuous Probability Density Function (PDF) [13]

of error for S_2 would be the tail of the S_2 PDF which lies to the left of the threshold [13].

Density estimation is a way of estimating the PDF of a process given a finite number of samples. The two non-parametric estimation techniques examined in this thesis are k-nearest neighbor and Parzen window techniques.

The k-nearest neighbor (k-NN) technique for estimating PDFs is based upon a volume created by enclosing the k nearest samples to a data point by a contour of constant distance [13, 18]. An example of this can be seen in Figure 17. If the volume is small, the points are densely packed and the probability of an outcome falling within this region is high. The PDF has a large value at this point. When the volume is large, the points are spread out and the probability of an outcome falling in this region is low. The PDF has a small value at this point.

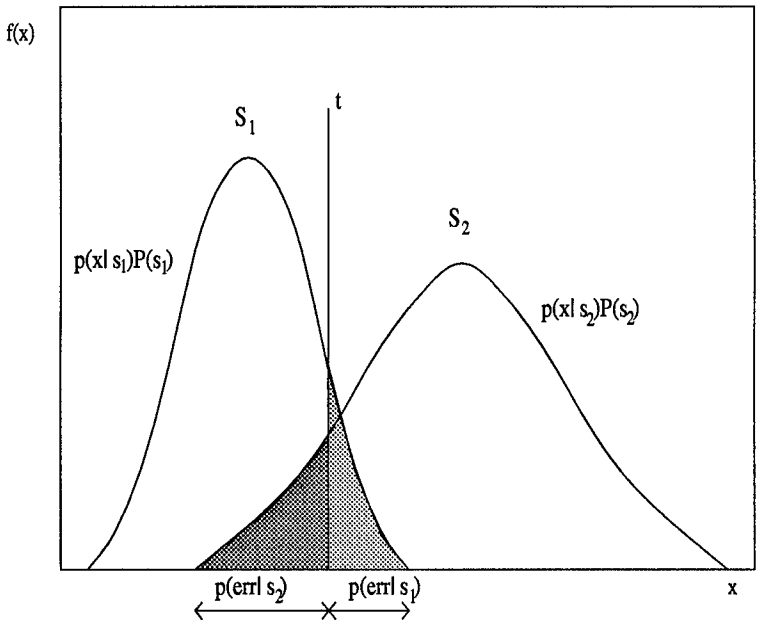


Figure 16. PDFs and Error Probabilities [13]

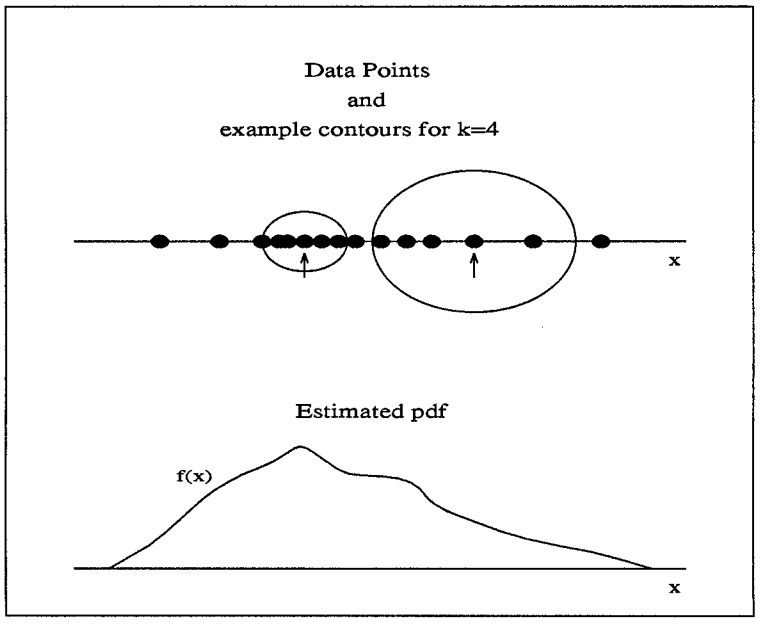


Figure 17. k-NN Density Estimation for k=4 [13]

The k-NN density estimation for a given class at point x is

$$\hat{p}(x) = \frac{k-1}{N\ V_k(x)} \tag{28}$$

where N is the total number of samples, k is the number of neighbors, and V_k is the volume of constant distance which encloses the k nearest neighbors [8, 13, 18].

To calculate the volume, a distance metric is chosen. The metric used here is the squared Mahalanobis distance described by

$$d^2(\bar{x}, \bar{y}) = (\bar{x} - \bar{y})^T \sum\nolimits_i^{-1} (\bar{x} - \bar{y}) \tag{29}$$

which uses the covariance matrix to scale each axis of the n-dimensional space proportional to the variance in each dimension [8, 18]. Note that $\sum_i^{-1}$ denotes the inverse covariance matrix.

For an n-dimensional space, a surface of constant Mahalanobis distance is a hyperellipsoid [8, 13, 18]. The volume of the hyperellipsoid is

$$V = V_d \left|\sum\right|^{1/2} r_{ik}(x) \tag{30}$$

where

$$r_{ik} = \sqrt{d_i^2(x, x_{k-NN}^{(i)})} \tag{31}$$

V_d is the volume of an n-dimensional unit radius hypersphere.

$$V_d = \begin{cases} \frac{\pi^{n/2}}{(n/2)!} & \text{n even} \\ \frac{2^n \pi^{(n-1)/2} \left(\frac{n-1}{2}\right)!}{n!} & \text{n odd} \end{cases} \tag{32}$$

These equations lead to the definition of the class ω_i PDF at a point x as

$$\hat{p}_i(x) = \frac{k-1}{N_i V_d |\sum_i|^{1/2} r_{ik}^n(x)} \tag{33}$$

The Parzen window technique for estimating PDFs consists of placing a small window function at every data point. When the areas of all the window functions are added together, the resulting function approximately estimates the density function. An example is shown in Figure 18.

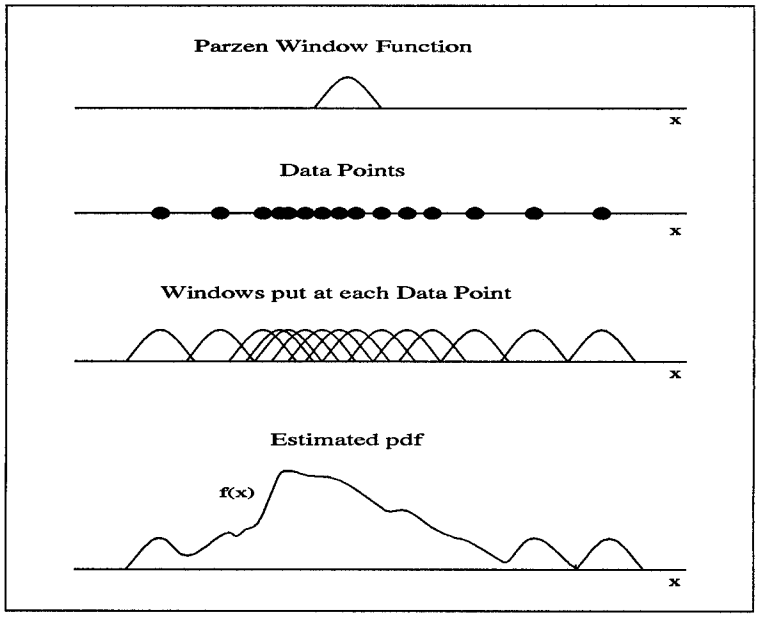


Figure 18. Parzen Window Density Estimation [13]

Mathematically, the Parzen estimate of the PDF at point x for one class of data in n-dimensional space is

$$\hat{p}(x) = \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{1}{h^n} k_i \left(\frac{x - x_j^{(i)}}{h} \right) \tag{34}$$

where $k(\cdot)$ is the kernel function, h is a parameter which controls the window spread, N is the number of samples, and x_j are the samples [8, 18].

The code used to find the error incurred through classification was obtained from Curtis Martin and modified to give the output information of interest to this thesis work [13, 18]. This code may be found in Appendix C.

Different Matlab files are used to put the data into a form suitable for use in the classifying code. These files define an up matrix, a down matrix, a hold matrix, the Parzen window size, and the number of nearest neighbors to be used in the classifying code. The up, down, and hold matrices are created so that the columns represent separate data points and the rows represent the n-tuple time partitions associated with those points. The two matrices do not have to be the same size, but they must contain the same number of rows. This determines the error probability for a given k or h value [13, 18].

In the original unmodified code, the data matrices, a range of values for h and k, and the leave-one-out option are passed to the pknn.m function. Then the following procedure is performed five times and the results are averaged together [13, 18].

- For each set, every fifth point is taken as a test sample and the rest of the points are used to create the covariance matrices.
- The sample sets and inverse covariance matrices are passed to the function compute_distances.m which computes the Mahalanobis distances between all the points, both inter-class and intra-class. The distance matrices formed are then returned to pknn.m.
- For each value of h, the resubstitution and leave-one-out discriminant values are calculated for the Parzen window technique. These values are passed to classify.m which calls min_error.m to find the resubstitution minimum error and threshold. This is done by determining how the two discriminant value sets overlap, varying a threshold over that region, and counting the errors. The resubstitution minimum error and threshold values are returned to classify.m and then the leave-one-out minimum error and threshold are calculated in a similar fashion. These error values are then returned to pknn.m.
- The same procedure is followed for the k-NN technique for each value of k.

- All of the error values are then passed back to the driver script file where plots can be created [13].

2.8 Bayesian Classifiers

Bayesian estimation models the parameters to be estimated as random variables with some (assumed) known *a priori* distribution [28]. The training samples are considered observations of the *a priori* information in order to approximate an *a posteriori* density. The training set is thus used to update the training set-conditioned density function of the unknown parameters [28].

The non-parametric k-NN classifier uses direct classification based upon the training set. The k-NN algorithm breaks the feature space R^n into k decision regions. A new vector presented to the algorithm is classified by its location relative to the decision regions. The decision boundaries are set for each problem with respect to the distance metric chosen. An example of this classifier is presented in Figure 19.

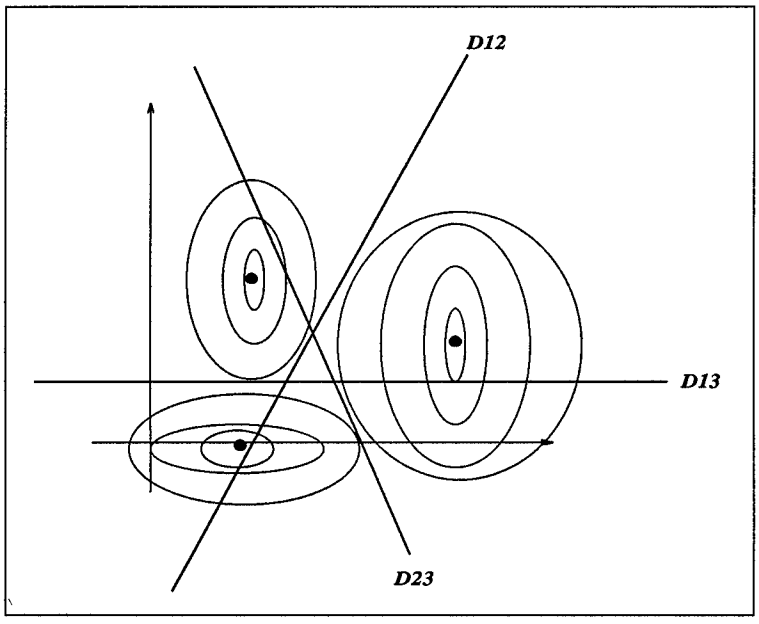


Figure 19. k-NN Algorithm Splitting the Space into k Decision Regions

Another way to classify the data points is to determine the k vectors nearest the test vector. The class of each training set is determined. The class of the test vector is determined by a majority vote rule of the k training vectors' classes [16]. For example, for a two class problem, a point which has 16 nearest neighbors 02 the class which has 9 or more examples. If no class has a majority, the class whose observations are closer to the point is chosen.

2.9 Summary

Chapter II reviews work done in a number of research areas which are being investigated in an attempt to improve the prediction capabilities of current embedology techniques. Nearest neighbors, spectral estimation, and neural networks, and non-parametric density estimation are all examined to provide the reader with a basic understanding of the theory behind the procedures being investigated in the following chapter. The next chapter examines data preparation through detrending, the use of the spectral components to provide a confidence measure of the embedology prediction, the use of nearest neighbors as the training set for a neural network, the use of different neural network architectures to aid in the classification of the prediction point, the Bayes Error Bounding using non-parametric density estimation for classification of the data, and fusion of the methods.

III. Algorithm Procedures for Time Series Prediction

3.1 Introduction

This chapter outlines the development of the improvements made to the Deterministic Versus Stochastics (DVS) algorithm proposed by Martin Casdagli in 1991 [1]. Section 3.2 examines the steps taken to prepare the data sets for use with the various algorithms. Section 3.3 examines the use of the data's phase to determine the accuracy of the prediction. Section 3.4 examines the use of the multilayer perceptron (MLP) and Radial Basis Function (RBF) architectures in prediction. The inputs for these networks are time series vectors of size m, the embedding space size, as determined by Taken's Rule and the fractal dimension of the data [32]. Section 3.5 examines the algorithm which uses the nearest neighbors, as determined by the Deterministic Versus Stochastics algorithm [1]. The nearest neighbors are the training set for a neural network, in order to determine if an accurate prediction can be made with this reduced data set. The use of this algorithm may also determine a new confidence measure of the prediction if a number of nearest neighbors with the same trajectories in state space can be found which consistently give a correct prediction. Section 3.6 examines the Bayes' Bounding probability curves in order to determine the prediction accuracy attainable using a given data set. Section 3.7 then examines a fusion algorithm of the aforementioned concepts in order to develop a robust prediction network.

3.2 Data Manipulation and Preparation

Time series data, as a rule, does not contain a lot of extraneous information which may be necessary for the proper use of an algorithm. In fact, most time series data simply contains the time signature, such as minutes, hours, days, ..., and the data samples themselves. In order to prepare the data for use in the algorithms, a number of parameters must first be determined.

Given a time series for local linear prediction, the data is assumed to be fractal, which implies that a correlation (fractal) dimension d has been determined for it [31]. This allows for the use of Taken's Rule and Sauer's Embedology Theorem, which have already been discussed in Section 2.2, in making a determination of the embedding dimension, m [27, 32].

Once the embedding dimension, m, is chosen, some number k of neighbors nearest the m-tuple closest to the prediction point must also be chosen. Both of these parameters are determined through the examination of several values for each. Those values which give the minimum amount of error are chosen.

In order to create a time series having locally identical statistics, the data set must be detrended [31]. Detrending is necessary in that many time series exhibit strong upward and downward trends. These trends may cause the final m-tuple, used as the point nearest the prediction point, to have a small number of neighbors. With such a small number of neighbors, the regression is rendered ineffective [31]. This trend action is strong in the financial data being examined here, which makes detrending a necessary action.

There is more than one way to detrend the data. The simplest, and most often seen in research, is that of first differences. The method of first differences approximates the first derivative of the data, y_n, by taking the difference of x_n and the previous point x_{n-1}. This method acts like a high pass filter, only letting low frequency components through.

$$y_n = x_n - x_{n-1} \tag{35}$$

For n=1, the data point y_0 is set to zero, to keep the time signatures accurate but not for use in any prediction method. The scoring of the detrended data, in order to relate it to the original data, is a signum function. When a data point from the detrended set is predicted, the sign of the prediction point y_n determines the

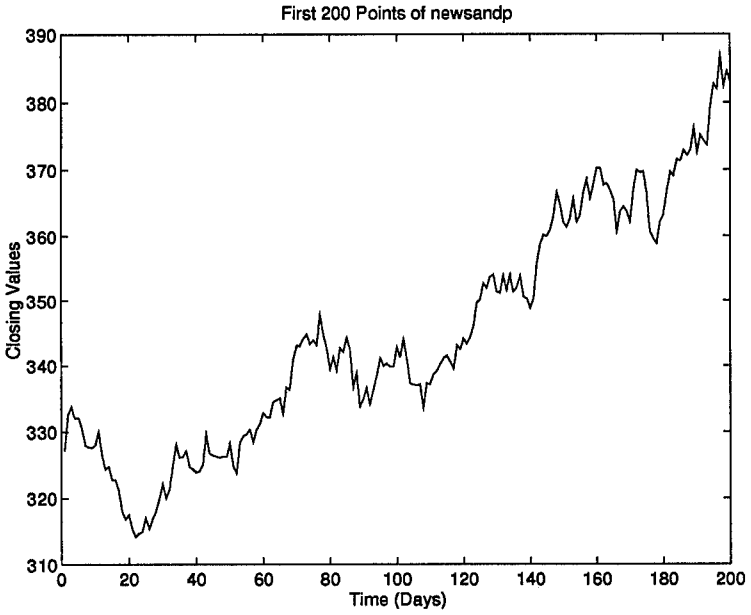


Figure 20. The S and P Data Set

movement of the original data set : up, down, or hold. For example, predicting point 900 of the detrended data set gives a negative number as the output from the algorithm. The negative number for point 900 of the detrended data set maps to point 900 of the original data set minus point 899 of the same set. Point 900 of the original data set is thus predicted to be the value of the detrended prediction point 900 less than point 899.

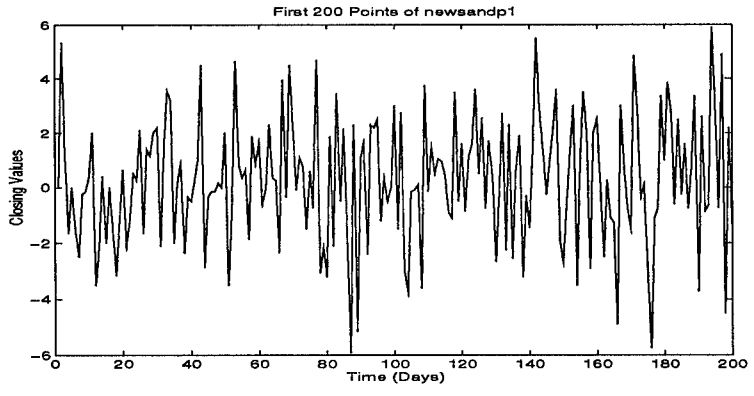


Figure 21. The S and P Data Set Detrended by First Differences

Another detrending method is a modification of first difference called First Ratios. Instead of subtracting x_{n-1} from x_n, x_n is divided by x_{n-1}. Again, this method acts like a high pass filter, with the added feature that it reduces the magnitude of the series.

$$y_n = \log(\frac{x_n}{x_{n-1}}) \tag{36}$$

The base-10 logarithm of the set is taken in order to prepare the data for scoring. The relative nearness of one point to the next insures the proportion of the detrend function is approximately one. Taking the logarithm of this set thus enables a positive-negative scoring technique since a detrend value of less than one, which corresponds to a larger previous number in the original data set, is negative. A detrend value of greater than one, denoting a larger prediction point in the original set, is now positive. This enables the use of the same scoring technique as the first differences set.

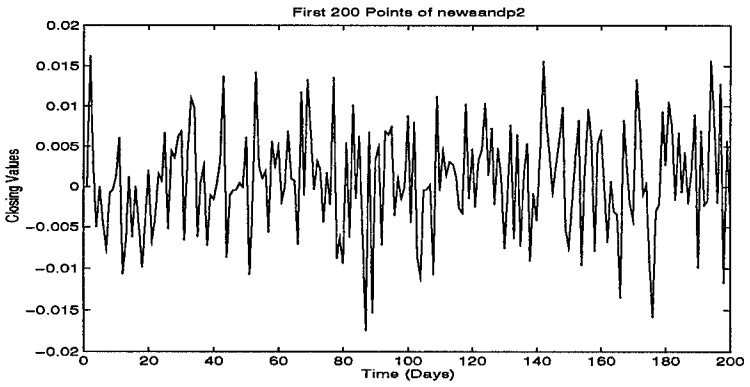


Figure 22. The S and P Data Set Detrended by First Ratios

A final detrend method examined is based upon edge detection techniques used in image processing called Image Ratios [23]. This method consists of the first difference divided by the addition of the points. Another high pass filter method, the data's magnitude is reduced even further than by First Ratios.

$$y_n = \frac{x_n - x_{n-1}}{x_n + x_{n-1}} \tag{37}$$

This method is successfully used to detect edges in images, and is theorized to do well in local linear prediction [21]. Scoring for this method is the same as the previous two methods. The prediction point y_n of the detrended data set is negative if the previous point of the original data set, x_{n-1}, is bigger than the original data sets x_n. Likewise, the prediction point y_n of the detrended data set is positive if the previous point of the original data set, x_{n-1}, is smaller than the original data sets x_n.

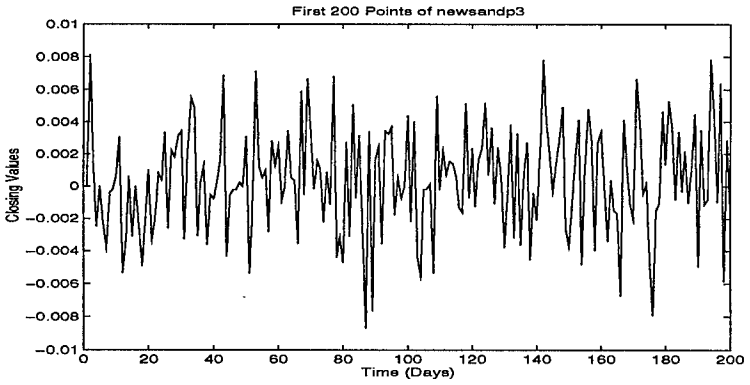


Figure 23. The S and P Data Set Detrended by Image Ratios

The different methods used to improve upon the Deterministic Versus Stochastics algorithm have varying requirements for data presentation. Although discussed in detail in each individual section, a brief overview of the data requirements is presented here. Detrending and setup of the data is done using the MATLAB package. These algorithms can be found in Appendix C.

For implementation with the Deterministic Versus Stochastics algorithm, as well as the Prony Method, the data set must be presented as a single vector of the data values leading up to the prediction point. Time signatures exist only as the vector's indices. Both of the data sets examined here, the S and P data set and the INFFC data set, are presented as several data vectors along with a single vector of time signature. These data sets may be reduced to single vectors in a variety of ways, such as using UNIX's AWK and SED functions [10]. Data manipulation schemes can be found in Appendix B. Once this is done, these sets can be detrended for use.

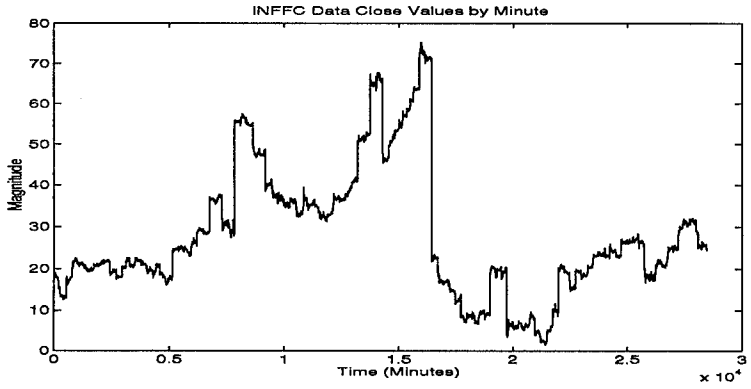


Figure 24. The INFFC Data Set

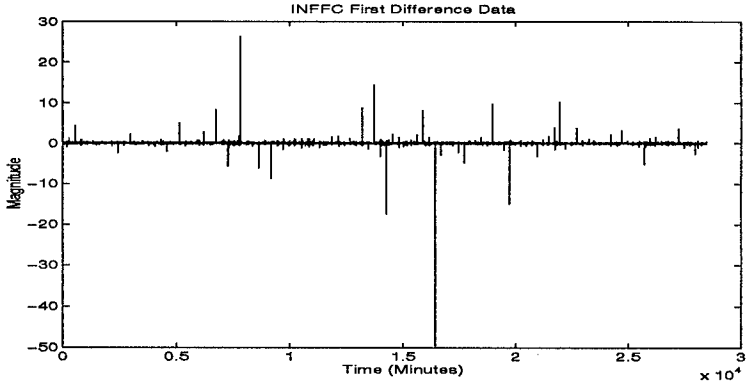


Figure 25. The INFFC Data Set Detrended by First Differences

The inputs to the neural network, whether using the multilayer perceptron or radial basis function architectures, are the time series values which make up the m-tuples of the data set, where m is the size of the embedding space of the data set. Since the neural network is used with detrended data, the data must be detrended and then broken up into component m-tuples before use.

The data must be prepared twice for the nearest neighbors calculations. The nearest neighbors are determined using the Deterministic Versus Stochastics algorithm, which means the data must be prepared, for size and detrending, as discussed previously. For use as the inputs to the neural network, however, the data must be broken up into vectors of length m, which determine the nearest neighbor m-tuples. This process needs to be performed upon the detrended data, which is already available from the Deterministic Versus Stochastics portion of the algorithm.

3.3 Prediction Methodology Using Casagli and Spectral Estimation

In an attempt to improve prediction accuracy, many researchers attempt to determine when their algorithms are the most practical. In many ways, it is just as important to know when the time series is most successfully predictable using a particular algorithm. This allows the attractor to be broken up into highly predictable and unpredictable areas. If an algorithm is known to fail in a certain area of the attractor, then efforts can be made to determine how to improve the algorithm specifically for that area. If it is possible to eliminate or ignore some of the predicted data, then knowing where the algorithm fails will effectively eliminate all prediction points located in the same portion of the attractor from contention.

Using these ideas, a hypothesis that the phase of the time series is able to determine whether the prediction is accurate was formed. The Fourier Series components are used to find out what the major phase components of the original time series are. Once the phases are determined, the rise and fall of the time series is compared to the rise and fall of the phase component. The accuracy of the predictions are

increased by only accepting the prediction points which follow the rise and fall of the phase components, when the two series are "in phase."

To test out the viability of this method, other methods of frequency analysis are examined to determine a likely replacement for the Fourier Series analysis. The method chosen to determine the time series' phase is the Least Squares (TLS) Prony Method. This method is the most accurate of the autoregressive (AR) modern spectral analysis techniques currently found in the field of signal processing [25]. This method is used to determine the spectral components of the time series based upon a window of data immediately proceeding the prediction area. Assuming that the phase components found in the window are representative of those found in the prediction area, the phase components are then used to define a second time series. Keeping the prediction points only when the phase's time series is "in phase" with the prediction series should hypothetically increase the accuracy of the prediction set.

To test this, a prediction set of data is first determined using Casdagli's Deterministic Versus Stochastics algorithm, which has been implemented here by Jim Stright [31]. Stright's code 0 the algorithm as presented in Section 2.3. The fitting set size N_f is set to one less than the desired prediction point. For example, to predict point 900, N_f is equal to 899. The test set size N_t is set to zero, in order to determine the delay vectors x_j from the fitting set. Set the value of the embedding dimension at a small value, such as 3 or 4. If the fractal dimension, d, of the data set is already known, set m equal to 2d+1, as determined by Taken's Rule [32].

Run the code using a C compiler. The output of the program is a series of error values for k, along with a prediction point. If m is determined using Taken's Rule, choose the value of k which gives the minimum error. If m is being chosen arbitrarily, m needs to be varied and the algorithm rerun. The m and the k values which give the minimum error are chosen. All prediction points prior to this point are ignored. Once m and k are chosen, the algorithm is rerun to determine the

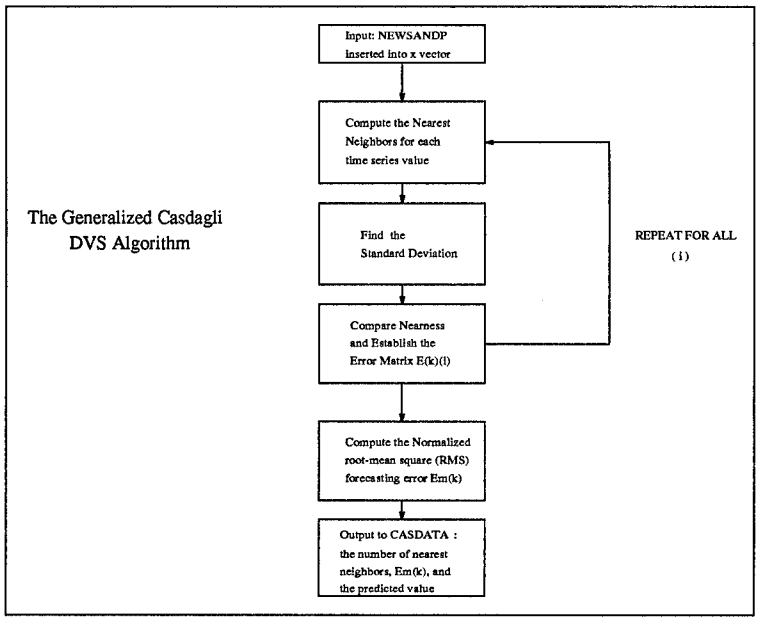


Figure 26. Block Diagram of the Casdagli DVS Algorithm

prediction point. Debate continues on the length of the prediction set before a new k must be chosen, although research done by Stright indicates that the use of a global k value is approximately as accurate as using local k values for equally sized test sets [31].

For use with Casdagli's algorithm, the Standard and Poor's data set is detrended using the three methods discussed in Section 3.2. This creates the files newsandp1, newsandp2, and newsandp3, which implement the detrending methods of First Differences, First Ratios, and Image Ratios respectively. These detrended sets are single vectors of data, in order to conform with Casdagli's algorithm's requirements, which are thus also usable with the TLS Prony Method.

The TLS Prony Method, as implemented by Sacchini, uses least squares approximation to determine a maximum likelihood estimate of the poles of the data. This method is able to achieve greater resolution than Fourier methods for the same number of points [25]. This allows the TLS Prony Method to work well with fewer data points than the Fourier Series.

The TLS Prony method needs to have a model number in order to make the pole estimates. The model number is the number of frequency spikes expected in the spectrum of the signal. This corresponds to twice the number of sinusoids present in the data, since the Fourier Transform of a sine wave consists of positive and negative frequency impulses. Overestimating the model is not a bad practice since the model presents the additional impulses with small amplitudes and large distances from the unit circle.

The assumption of how many poles to keep needs to be made. Through experience, the choice of the number of poles to keep varies greatly from one problem to the next. In this case, after running several examples, the best criterion for elimination is based upon the amplitude coefficients. Keeping those amplitude coefficients which are of similar magnitude eliminates most of the spurious poles. This technique keeps out small fluctuations in the data set, which do not improve the prediction accuracy.

The size of the data window can be varied in order to determine if the frequency components of the data change. This indicates that trends can occur over different time periods of data. Thus, a frequency which occurs over a hundred point window may not be observable when using a twenty point window. This change in the data size can lead to an optimal data window for phase determination.

This windowing assumption leads to several window sizes being used for spectral estimation. These windows are taken from the data with the forward window edge falling on the last point of the fitting set. This analysis assumes that the phase components found within the windows are representative of the local phase phenomena in a periodic fashion, which thus allows the phase components to be applied to the prediction set.

The TLS Prony Method is ran to provide the pole estimates, amplitude coefficients, and energy for twenty point, forty point, and one hundred point windows. This means, for example, that if the point being predicted occurs on the set 900 through 919, then the Prony Method uses the data vectors from 880 through 899,

860 through 899, and 800 through 899, respectively. The parameters are listed out according to the energy ranking of the poles, with the pole with the highest energy content being listed first. Once listed, the poles are analyzed for applicability to the current problem. The poles whose distance from the unit circle is too great are eliminated. Those poles whose amplitude coefficients are a magnitude of size, or more, smaller than the largest pole are also eliminated.

Once this is done, the amplitude coefficients are used as the multiples of the sinusoids defined by the poles.

$$phasedata(n) = a_1 * sin(2\pi f_1 n) + a_2 * sin(2\pi f_2 n) + \ldots \tag{38}$$

This model is then used to create a data set whose size is equal to the number of predictions taken using the Deterministic Versus Stochastic algorithm. These points are then analyzed to determine the phase movement, using zero as a starting point.

In the case of the Standard and Poors and INFFC sets, the setup is identical for all three sets. First, the size of the testing sets is set at twenty points. The Deterministic Versus Stochastics algorithm is then used to predict the twenty points for each set. These predicted points are analyzed to determine whether each prediction point is higher or lower than the previously predicted point. In order to determine the prediction set's true accuracy, compare each predicted point to the previous actual point. Once this is done for all twenty points of the test set, the TLS Prony Method is used.

The TLS Prony Method is setup for three sets of phase determination. The first set uses the 20 point data window, the second uses the 40 point data window, and the third uses the 100 point data window. The TLS Prony Method is used, with the output of each set being recorded in a data file. Each of these data files is examined to determine the relevant poles. Once the poles are picked for each set, the

pole's phase and amplitude coefficient are put into the model in order to determine the phase data set.

Each phase data set is placed next to the corresponding Casdagli prediction set. If the movements are "in phase", such as both points move upward, then the prediction point is kept. If the points are "out of phase", where one point moves up while the other moves down, then the prediction is disregarded. This is done for all twenty points of each test set. The relevant prediction points are then analyzed to find the reduced set's true accuracy, by comparing the predicted points to the actual points for the reduced set. If the hypothesis is true, the reduced set should have a higher correct prediction accuracy than the total set.

3.4 Prediction Methodology Using Artificial Neural Networks

The processing methods commonly used today, such as Casdagli's Deterministic Versus Stochastic and Sauer's Embedology algorithms, employ the use of linear hyperplanes, fit to the nearest neighbors through regression techniques, to determine the likely trajectory of the m-tuple nearest the prediction point [1, 27]. This estimation technique allows the prediction point to be determined through projection of the trajectory on to the hyperplane.

In order to improve upon the process, researchers have endeavored to place nonlinear surfaces through the nearest neighbors in order to determine the prediction point. In most cases, this method has been shown to yield very little, if any, improvement [27]. Subsequently, research into this method is currently minimal. This thesis will try to prove the validity of using a nonlinear surface.

For the S and P data set, the embedding space size, m, is chosen through the use of the Casdagli algorithm. This m is used to partition the data into its component m-tuples. For this 1100 point data set, m is determined to be 7. This allows the data set to be broken up into 1093 7-tuples of data, with the first 7-tuple corresponding to the 8th point of the original index. Breaking the data up into a

training set of the first 850 training vectors, 243 vectors are left for testing. Testing is done on ten 20 point sets and one 240 point set.

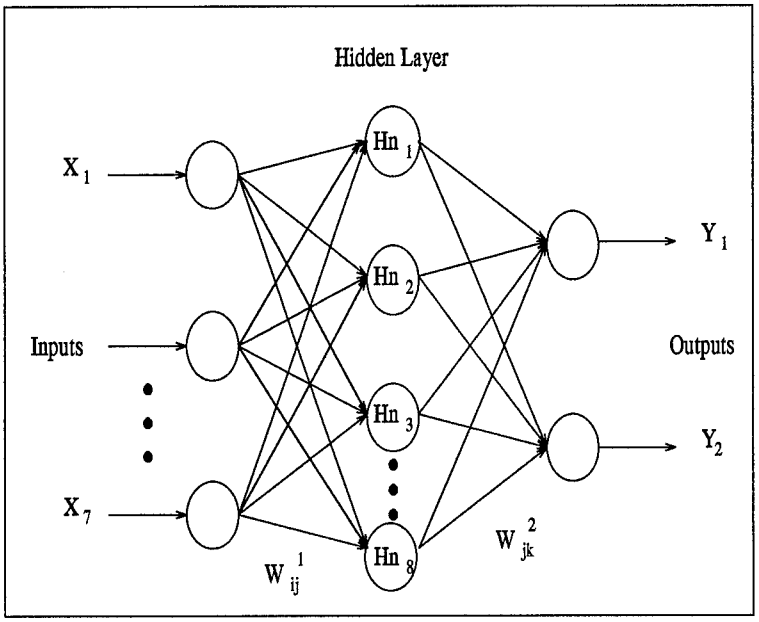


Figure 27. Multilayer Perceptron Setup for the SandP Data Set

To determine the architecture for the multilayer perceptron, the node equations 24 are applied.

$$85 \leq 8 * Hn + 2 * Hn \tag{39}$$

This shows that the maximum number of hidden nodes usable without memorization occurring is eight. The data set is reasonably sized, so the step size is set to 0.1, with a varying momentum starting at 0.5. The multilayer perceptron is ready for use. To determine the number of cluster centers to use, K-means clustering is done and the training error is examined. This determines that 16 clusters should be used for this data set. The Radial Basis Function network is now ready to use.

For the INFFC data set, the embedding space size, m, is determined to be 3. The 28465 point sample-and-hold data set partitions into 28462 3-tuples. these are evenly divided into training and testing sets of 14231 vectors each. To determine the

multilayer perceptron architecture, the node equations 24 are applied.

$$1423 \leq 4 * Hn + 3 * Hn \tag{40}$$

This determine that the maximum number of hidden nodes is 203. The data set is large, so the step size is set to 0.01 with a varying momentum starting at 0.05. Moving on to the Radial Basis Function, the K-means clustering algorithm determines that 32 is the optimal number of cluster centers for this data set. Both architectures are now ready for use.

A top down methodology to determine the optimal number of hidden nodes and number of training epochs is employed. The initial number of training epochs is set at one hundred, with a variable number of hidden nodes ranging from the maximum number down. The number of hidden nodes which determines the smallest error for the large testing set is chosen. Once the architecture is set, the number of training epochs is varied until the number of epochs which determines the least error for the large testing set is determined. Once the number of training epochs is determined, the neural network is ready for testing using the smaller test sets.

3.5 Prediction Methodology Using Nearest Neighbors

After looking at neural processing, a question arises concerning the locality of the prediction within the data set. The time series is constantly changing with time, which naturally leads to the conclusion that the parameters of the data set are also constantly changing. This has been shown to be true, but of little consequence to the prediction accuracy [31]. Choosing a global set of parameters accomplishes the same prediction accuracy as locally choosing them.

What of the locality of the prediction algorithms? Is it possible to train a neural network predictor locally to achieve the same level of accuracy as when training on the entire training set? These questions have led to the use of the nearest neigh-

bors for the m-tuple closest to the prediction point as the training set for a neural network. The nearest neighbors, determined by the Deterministic Versus Stochastics algorithm, are found using Euclidean distance as a metric. The k-nearest neighbors are the k closest points in m-space to the last point of the fitting set, N_f. This gives k vectors of m-dimensional size.

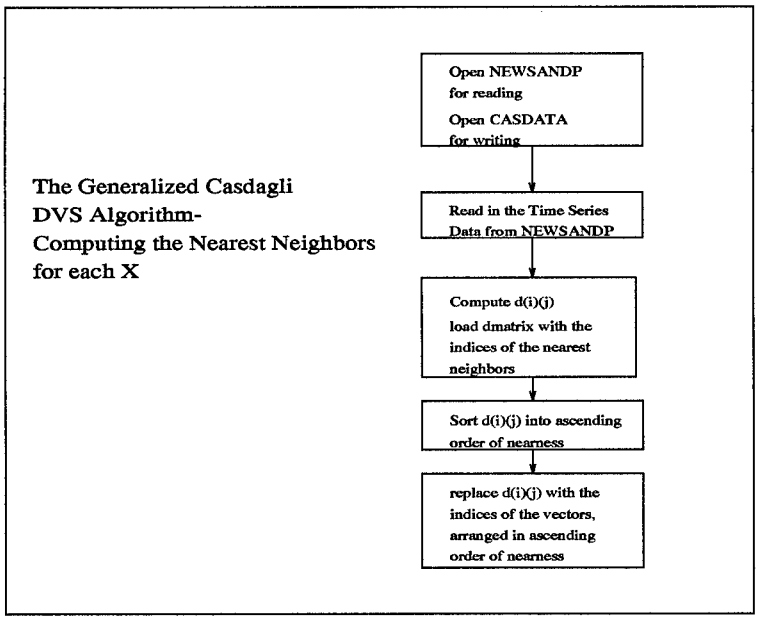


Figure 28. Block Diagram of Nearest Neighbor Algorithm using DVS Algorithm

The Deterministic Versus Stochastics algorithm uses the nearest neighbors in order to fit the linear hyperplane used for prediction (See Section 2.3). The rest of the training set, the fitting vectors, are relatively useless once the covariance matrix is determined. This verifies the importance of the nearest neighbors for the actual prediction and reiterates the question of whether a local training set can match the accuracy of the entire training set.

Supposing that a nonlinear surface better generalizes the data's underlying dynamics, the use of the nearest neighbors as the training set for a neural network becomes a necessary exercise. The use of the small, local training set allows for

shorter training times, as well as focusing the neural network's adaptation abilities on the most 0 training data with regard to the prediction.

As with the general neural adaptation techniques presented in Section 3.4, both the multilayer perceptron and the Radial Basis Function architectures are used. The same requirements apply, except that the number of weights need not necessarily determine the number of nodes required. The neural network's memorization of the training data is not the critical error it is under normal circumstances. The nearest neighbors form a surface based upon their immediate presences in state space. Memorizing the data will simply fit the nonlinear surface more precisely to the points present. Since these points are conjectured to carry enough information to formulate a prediction with an accuracy equivalent to that of the Deterministic Versus Stochastics algorithm, the resulting surface is more than sufficient for the task at hand.

In order to create the desired algorithm, a number of nodes must be decided upon. At least one hidden node is needed to fit a nonlinear surface to the training data. The fewer hidden nodes, the longer it takes the neural network to memorize the data. In order to fully determine the best architecture, a number of hidden nodes must be started upon and then reduced in a top-down fashion. This causes the neural network to better generalize the underlying dynamics of the data as the architecture gets smaller. The architecture which provides the smallest error is chosen.

The neural networks are set up as in Section 3.4. For the Standard and Poors data set, the number of nearest neighbors, k, is determined to be 16. The embedding dimension size, m, is 7. This gives a training set of sixteen 7-tuples. The test set is a single 7-tuple. The step size needs to be large since the training set is so small. The step size is set to 0.2, with a variable momentum starting at 0.5. The Radial Basis Function does not receive clustered information in this case, but rather works based upon the actual training vectors as the cluster centers.

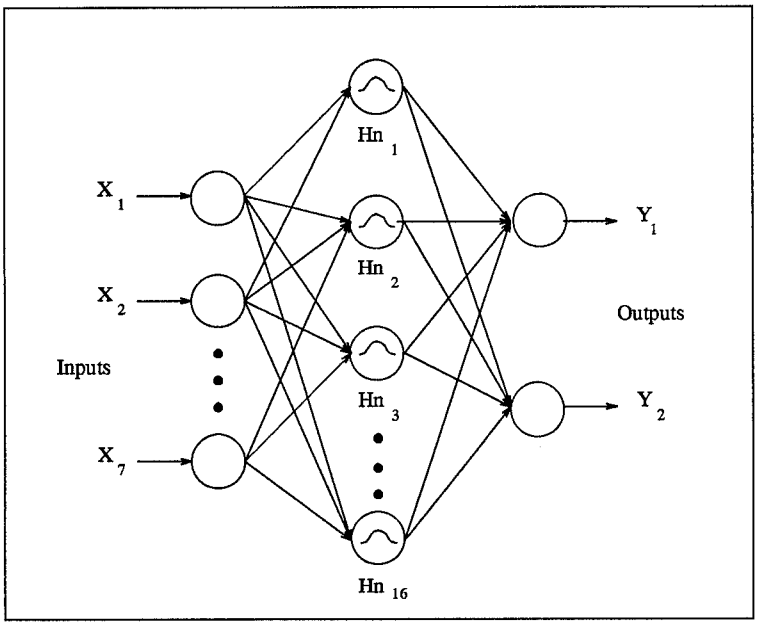


Figure 29. Radial Basis Function Setup for the SandP Data Set

For the INFFC data, k is set at 5 to limit the computation time. The embedding dimension, m, is 3. The training set is made up of 3-tuples. The step size is again 0.1 with a variable momentum starting at 0.5. The Radial Basis Function again works with the cluster centers set at the actual training data vectors.

3.6 Non-Parametric Bayes Error Estimation

Curtis Martin's code is used to find the Bayes Bound on the data's error probability [18]. The data set needs to be organized properly for the algorithm first. The data set needs to be split up into smaller data sets for each class. The code is written for a two class problem, although it can be applied to larger problems by breaking them up into a set of two class problems. Each class needs to be organized with each row defining a different feature. For the data sets examined in this thesis, the m features are the time series points which make up the m-tuples. Thus a data set with an m of 7 is broken up into component data sets with 7 rows each.

Remember that the number of data points in each class set need not be equal, but that the number of features must be the same for each class.

The algorithms used to create the two class sets are MATLAB functions. In order to better characterize the data set, the parameter m is varied along with k and h. The variation of these parameters allows for the combination which determines the best error bounds to be found. Since a range of k and h is inherent in the algorithm, m is the parameter which needs to be varied. The parameter m is varied by changing the embedding space of the data. Since the features are made up of the m-tuple points, varying m varies the number of features.

For the S and P data set, the parameter m is varied from 1 to 10. This range is based upon the determination of m using the DVS algorithm. Since the DVS algorithm gives an m of 7 for this data set, the range 1 to 10 is deemed sufficient to complete the experiment without inducing too much computational complexity. The data sets are broken up into classes of up and down movements. This two class set needs a single run of the Martin code to determine the Bayes Error Bound.

For the INFFC data set, the parameter m is varied from 1 to 3. This satisfies the fractal dimension determination of the embedding space size, m, as 3. Since the Martin code is set up for a two class problem, this three class problem is broken up into its two class problem components. The data sets three classes are up, down, and no movement (hold). The three component data sets are the up-down set, the up-hold set, and the down-hold set. The fusion of these three sets characterize the error probability bounds of one m-tuple data set.

3.7 Bayes Classifiers

The k-NN algorithm is 0 using the LNKnet software package [16]. LNKnet's k-NN classifier trains by storing all of the training vectors presented to it. During testing, the k stored patterns closest to the test vector are found using the Euclidean

distance metric. A vote is taken amongst the k neighbors and the class which occurs the most is assigned to the test pattern [16].

The algorithm queries the user for the value of k. The k value can be obtained from the DVS algorithm (See Section 2.3). The k value can also be determined from the PDF estimation using Martin's code. The k value which gives the least error is determined twice in the Martin Code, using two threshold options. For these experiments, each of Martin's k values are used, along with the k determined using the DVS algorithm, in order to see which k determines the smallest error.

For the S and P data set, the k value determined by the DVS algorithm is 16. The k values determined by Martin's code are 25 and 42, as determined by threshold options 1 and 2 respectively. These three k values are used as the input to LNKnet's k-NN algorithm. For the INFFC data set, the k value determined by the DVS algorithm is 5. The k values determined by Martin's code are 7 and 9. These three k values are then used as the input to LNKnet.

3.8 Fusion of the Algorithms

Fusion of the various algorithms is done to create a more robust prediction network. Fusion is only advantageous if the different algorithms make incorrect predictions based upon different criteria. In other words, if an algorithm "loses" when another algorithm "wins", a fusion methodology can be formed in order to form an optimized prediction set based upon the two algorithms.

In reality, a fusion algorithm needs an odd number of "voters", i.e. prediction algorithms, in order to make a decision for a two class problem upon the collective prediction. With an odd number of algorithms, an even majority prediction determines the fusion prediction. The supposition is that the fusion algorithm attains a lower probability of error than that obtained by any one class.

For the S and P and INFFC data sets, the algorithms are compared pointwise in order to determine which prediction points they incorrectly classify. If the algorithms

all "miss" the same points, then a fusion method is ineffective, since the combination of the algorithms still incorrectly classifies the same points. If the algorithms "miss" different points, however, taking a majority vote across the algorithms may decrease the probability of error.

Probabilistic classification can also be used as a fusion method. Colombi has shown that postprobabilities may be used to combine, or fuse, various classifiers as well as features themselves [5]. Two different techniques are derived to use the postprobabilities of the various algorithms. the goal is to choose the maximum likely class i from the v classifiers.

The postprobabilities for each algorithm need to be derived for each class. For the DVS algorithm, the post probabilities are found by examining the relative probabilities of the nearest neighbors for each point. For instance, with 16 nearest neighbors, four points are defined as up and twelve points are defined as down. The relative probability for an up movement is 25 percent and for the down movement is 75 percent. The postprobabilities for the nearest neighbor neural adaptation, neural networks, and k-NN algorithms are all found using the LNKnet error files. the postprobabilities for each class are the values of the output nodes for these algorithms.

The first algorithm fuses the postprobabilities over a varying number of classifiers. This algorithm, the average fusion method, is defined by equation 41.

$$P_A(i) = \frac{1}{V} \sum_{v=1}^{V} p_v(i) \tag{41}$$

where $P_A(i)$ is the fusion probability of class i, $p_v(i)$ is the postprobability of class i for classifier v, and V is the number of classifiers used. The maximum $P_A(i)$ is the fusion algorithm's choice for the point's class.

For the S and P data set, the classifiers used are the DVS, the k-NN with k=16, the neural network with sets based on random seed 0 and 2, and the MLP nearest

neighbor neural adaptor with k=16. There are two classes, down = 0 and up = 1, with variable V based upon the number of classifiers used. For the INFFC data set, the classifiers used are the DVS, the k-NN with k=5, and the RBF nearest neighbor neural adaptor with k=19. There are three classes, down = 0, up = 1, and hold = 2, with variable V based upon the number of classifiers used.

The second algorithm also fuses the possibilities over the classifiers. This algorithm, the weighted probabilities method, is defined by the equation 42

$$P_W(i) = \sum_{v=1}^{V} W_v p_v(i) \tag{42}$$

where $P_W(i)$ is the fusion probability of class i, W_v is the weight associated with classifier v, and $p_v(i)$ is the postprobability of class i for classifier v, and V is the number of classifiers used. The weights are based upon the relative success of the algorithms. The better the prediction accuracy of the specified algorithm, the larger the corresponding W_v.

The same algorithm choices for the S and P and INFFC data sets exist as those used with the average fusion algorithm. The best combination of the algorithms derived for both data sets using the average fusion method is used, with the weights based upon the relative accuracies of the algorithms. The weights are varied until the best prediction accuracy is found.

3.9 Statistical Significance of the Algorithms

A determination of the statistical significance of the different algorithms when compared to the DVS algorithm is done to determine if the methods have different average prediction rates than the DVS algorithm. This is done through the use of the t-score. The t-score is computed as

$$t = \frac{\frac{1}{n}\sum_{i=1}^{n}(d[i] - e[i])}{\frac{s_d}{\sqrt{n}}} \tag{43}$$

where i= 1,...,10, d[i] and e[i] are results for test set i with the DVS and an experimental algorithm, s_d is the sample standard deviation of the paired differences, and n is the number of test pairs [29]. In these experiments, the paired differences have a t distribution with nine degrees of freedom. Under these conditions, a t-score with a magnitude greater than 1.833 indicates an experimental algorithm which is statistically different than the DVS algorithm [29]. The confidence interval for each algorithm is also found using the following formula

$$\bar{x} \pm t_{\frac{\alpha}{2}} \frac{S}{\sqrt{n}} \tag{44}$$

where $\bar{x}$ is the average prediction error, $t_{\frac{\alpha}{2}}$ is the table t-score, S is the standard deviation, and n is the number of test sets.

3.10 Summary

Chapter III reviews the hypotheses and algorithms used to explore the improvement techniques for Casdagli's Deterministic Versus Stochastic algorithm. Spectral estimators, neural networks, nearest neighbors, Bayes Error Bounding, Bayes Classifier, and fusion are all examined to show the procedures behind each experiment conducted for this thesis. Chapter IV examines the results of each experiment in detail.

IV. Results and Discussion

4.1 Introduction

This Chapter presents the results of the experiments in Chapter III. Section 2 examines the use of the TLS Prony Method as an acceptance method for the DVS algorithm. Section 3 examines the use of a neural network as a classifier while Section 4 examines the use of the DVS nearest neighbors as the training set for a neural network. Section 5 examines the PDF estimation of the data for error boundary calculation. Section 6 examines Bayes classification using the k-NN algorithm. Lastly, Section 7 examines several fusion methods of classification based upon the previous methods.

4.2 Casdagli's DVS Predictions

In order to determine how well the methods perform, the accuracy of the Casdagli DVS algorithm must be determined for the test sets. The S and P test group consists of the raw data detrended by First Difference, First Ratio, and Image Ratio methods as described in Section 3.2. Each detrend set consists of 10 twenty point data sets. The INFFC test group is made up of the raw close data. The test group consists of 10 twenty point test sets.

The S and P two class test group is used to determine which detrend method is used throughout the remaining research. For each detrend method, the prediction accuracy for each twenty point test set is calculated. The individual test set accuracies are then averaged to provide an overall prediction accuracy for the detrend method. Once the average prediction accuracy for all three detrend methods is computed, they are compared in order to chose a detrend method.

All of the average test accuracies are within half a percent of each other. Examining the behavior of the individual test sets for each detrend group, both the First Ratio and Image Ratio detrend sets have relatively consistent prediction

Data Set	Prediction Accuracy	Prediction Accuracy	Prediction Accuracy
	First Differences	First Ratios	Image Ratios
	(%)	(%)	(%)
860	65	55	50
880	50	55	55
900	55	55	55
920	40	55	55
940	60	55	55
960	55	55	55
1000	30	50	50
1020	55	50	50
1040	60	55	55
1060	45	35	35
AVG	51.50	52.00	51.50

Table 3. Casdagli DVS Predictions on S and P Data, k=16

Data Set	Prediction Accuracy	Prediction Accuracy	Prediction Accuracy
	k=5	k=19	k=110
	(%)	(%)	(%)
25000	40	40	45
25100	45	55	50
25500	20	15	25
25600	25	35	30
26000	50	30	30
26100	30	15	35
26500	30	45	30
26600	40	25	30
27000	35	15	35
27000	25	40	40
AVG	34.00	31.50	35.00

Table 4. Casdagli DVS Predictions of INFFC Data

accuracies, with 7 and 6 test sets respectively which are 55 percent accurate. The First Differences detrend set has a larger range of values, from 30 percent for test set 1000 to 65 percent for test set 860. Since the average test accuracies of the three sets are so similar, the First Differences detrend method is chosen because of its large test accuracy range and because it is the better known detrend set in this field.

The INFFC three class test group is not detrended. The Stright Code which implements the DVS algorithm can not function properly with this data set due to the large number of data points. The code has been revised to use the INFFC data, but cannot run with the detrended data because of the current decomposition method employed by the algorithm. The Lower Upper Decomposition (LUD) method currently used does not work with long strings of zeros, which occur for hold values. A Singular Value Decomposition (SVD) algorithm needs to be employed to be able to work correctly. The raw close data is used, with varying k values.

In order to choose the optimal k value, the test accuracies must be examined in much the same way as for the S and P data. The individual test set accuracies are determined, which enables the average test set accuracy for each k to be found. Since the data is found to be stochastic by the DVS algorithm, a reduction in root-mean-square error occurs as k approaches infinity [1]. The k values are thus chosen at various intervals in order to test the theory, with the first k value, 5, being determined by the Bayes Error Probability. The average test set accuracies vary, although the largest k value determined the best prediction accuracy. The k value determined by the Martin Code k-nearest neighbor algorithm is a close second, being the best k value less than 50.

4.3 Casdagli and Spectral Estimation

The TLS Prony Spectral Estimator is used to evaluate whether the DVS prediction is in phase with the two largest spectral components of the data set. Three different data windows are examined: 20, 40, and 100. In one set of predictions,

the spectral components are determined from the undetrended data, while for the other set of predictions the spectral estimators are determined from detrended data. Only the first differences detrended data is presented since the other two detrend sets do not have large enough magnitudes to determine the amplitude coefficients for the spectral estimators. In this case, no single component dominates since all of the magnitudes of the poles are approximately zero.

The results of the spectral estimation algorithm are presented with respect to the type of data set which determines the phase, raw or detrended, and window size. The prediction accuracy for each twenty point test set is presented, as well as an average prediction accuracy for the ten test sets. The rejection rate is presented for the individual test sets and for the entire set, where rejection is defined as not using a prediction point because it is out of phase with the spectral components. This allows for comparison with Casdagli's DVS algorithm on the individual test level as well as on the average. If there is no value for the prediction accuracy, no single spectral component dominates and all the points for that set are rejected.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	75.00	60.00
880	60.00	50.00
900	75.00	60.00
920	44.44	55.00
940	46.15	35.00
960	50.00	70.00
1000	36.36	45.00
1020	66.67	40.00
1040	37.50	60.00
1060	55.56	55.00
AVG	54.67	53.00

Table 5. DVS and Phase Prediction on S and P: Window - 20 pts.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	75.00	40.00
880	54.55	45.00
900	54.55	45.00
920	20.00	50.00
940	46.15	35.00
960	61.54	35.00
1000	35.71	30.00
1020	42.86	65.00
1040	75.00	40.00
1060	63.64	45.00
AVG	52.90	43.00

Table 6. DVS and Phase Prediction on S and P: Window - 40 pts.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	75.00	40.00
880	60.00	75.00
900	50.00	50.00
920	18.18	45.00
940	-	100.00
960	-	100.00
1000	33.33	40.00
1020	42.86	65.00
1040	72.73	45.00
1060	27.27	45.00
AVG	47.42	60.50

Table 7. DVS and Phase Prediction on S and P: Window - 100 pts.

Examining the S and P data set, two of the window sizes for the raw data phase set improve upon Casdagli's DVS prediction accuracies on the average, although not by a large amount. Individual test sets vary in their results, with some reporting impressive improvements while others show significant loss. Notice that the 860 test set in all of the windows is well above the DVS accuracy for the set. The 940 test set is significantly lower for the 20 and 40 point windows, and does not exist for the 100 point window at all, where the phase is determined to be 0. The determination of the best phase set of the three different windows depends upon the criterion of the user. If obtaining the best prediction accuracy supersedes the number of points actually tested, the twenty point window is chosen because the average prediction accuracy is approximately two percent more accurate than its competitor. If the amount of points being tested is more important than the exact prediction accuracy, the forty point window is chosen because it accepts ten percent more points than its competitor.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	57.00	65.00
880	42.86	65.00
900	54.54	45.00
920	44.44	55.00
940	72.00	45.00
960	54.55	45.00
1000	21.43	30.00
1020	72.73	45.00
1040	60.00	50.00
1060	50.00	50.00
AVG	45.68	49.50

Table 8. DVS and Phase Prediction on First Difference S and P: Window - 20 pts.

For the detrended data set, only the 100 point window's prediction accuracy surpasses the prediction accuracy of the DVS algorithm, and then only by a small

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	81.81	45.00
880	22.22	55.00
900	60.00	50.00
920	38.46	35.00
940	50.00	40.00
960	40.00	50.00
1000	30.77	35.00
1020	63.64	45.00
1040	72.73	45.00
1060	50.00	40.00
AVG	50.96	44.00

Table 9. DVS and Phase Prediction on First Difference S and P: Window - 40 pts.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
860	-	100.00
880	22.22	55.00
900	66.67	55.00
920	33.33	55.00
940	80.00	50.00
960	-	100.00
1000	28.57	30.00
1020	71.43	65.00
1040	66.67	55.00
1060	-	100.00
AVG	52.70	66.50

Table 10. DVS and Phase Prediction on First Difference S and P: Window - 100 pts.

amount. In essence, running the DVS algorithm is preferable to using the phase acceptance algorithm since the prediction accuracy is approximately one percent lower without any rejection. In fact, none of the accuracy rates for either the raw or detrended phase sets surpasses the DVS accuracy rate by a large amount, especially with respect to the rejection rate.

Examining the INFFC data set, three different data windows are examined: 20, 40, and 100. The spectral components are determined from the undetrended data, in order to compare the results to the DVS prediction accuracy. The prediction accuracy and rejection rate are presented for the individual test sets as well as for the entire test group.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
25000	25.00	80.00
25100	71.00	90.00
25500	11.00	55.00
25600	33.33	55.00
26000	62.00	60.00
26100	14.00	65.00
26500	25.00	80.00
26600	14.00	65.00
27000	14.00	65.00
27100	00.00	80.00
AVG	26.93	69.50

Table 11. DVS and Phase Prediction on INFFC: Window - 20 pts.

Although none of the average test set accuracies for the three windows reach the prediction accuracy of the DVS algorithm, the 40 point window comes the closest with a 33.17 percent accuracy. The range of prediction accuracies amongst the window sets vary, with the smallest range belonging to the 40 point window set. The rejection rates across the windows are fairly constant on average, ranging from 66 to 70 percent.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
25000	00.00	50.00
25100	43.00	65.00
25500	33.33	55.00
25600	27.00	45.00
26000	56.00	55.00
26100	29.00	65.00
26500	43.00	65.00
26600	17.00	70.00
27000	50.00	80.00
27100	33.33	85.00
AVG	33.17	63.50

Table 12. DVS and Phase Prediction on INFFC: Window - 40 pts.

Data Set	Prediction Accuracy using Phase (%)	Rejection Rate using Phase (%)
25000	00.00	75.00
25100	17.00	70.00
25500	00.00	70.00
25600	10.00	50.00
26000	40.00	50.00
26100	00.00	60.00
26500	17.00	70.00
26600	60.00	75.00
27000	20.00	75.00
27100	17.00	70.00
AVG	18.10	66.50

Table 13. DVS and Phase Prediction on INFFC: Window - 100 pts.

The individual test accuracies are often disparate from the DVS test accuracies. For example, data set 26000 has a DVS accuracy of 50 percent. The 20 point window accuracy for 26000 is 62 percent, the 40 point window accuracy is 56 percent, and the 100 point window accuracy is 40 percent. The window test accuracies are mostly above the DVS accuracy in this case. For data set 25000, however, the DVS accuracy is 45 percent. The 20 point window has an accuracy of 25 percent, while both the 40 point and 100 point windows have 0 percent test accuracies. In this case, all of the window test accuracies are below the DVS accuracy. In most cases, however, the test set accuracies vary in respect to the DVS accuracy.

4.4 Casdagli and Neural Networks

The use of a multilayer perceptron neural network with the parameters calculated by the DVS algorithm allows the use of a nonlinear surface to fit the points. The main concerns presented by this algorithm are the architecture size and number of training epochs. The architecture size and training epochs are chosen in the combination which determines the lowest test error. A top down approach to finding these variables is taken in this thesis.

For the S and P data set, the number of inputs is 7, determine by the embedding dimension obtained from the DVS algorithm. The number of outputs is 2, enough to classify the data as up and down. The number of middle nodes is varied, ranging from the maximum number, determined to be 8, down. Using the 200 point test set, the test errors are taken after 100 training epochs. The lowest test error is obtained with 4 middle nodes.

With the architecture determined, the number of training epochs is varied in order to minimize the test error. Examining the training errors for the 7,4,2 architecture, learning is completed before the original 100 epochs is complete. The number of epochs is thus varied from 100 down. Using the 200 point test set, the lowest error is obtained with 51 training epochs.

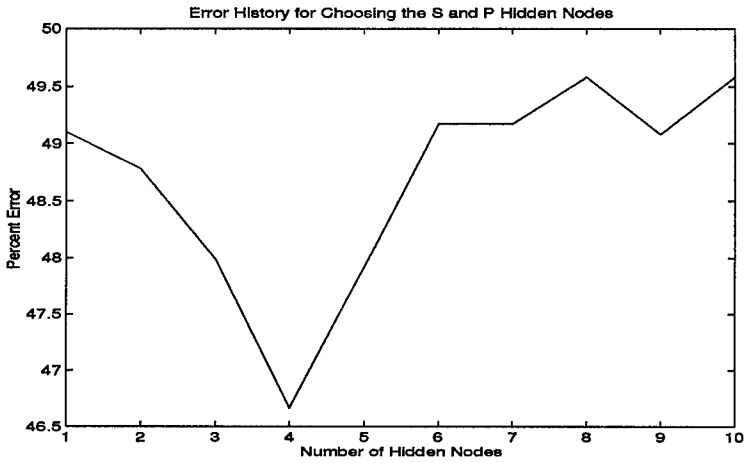


Figure 30. Testing Error History Plot of the 200 Point Test Set to Determine the Number of Hidden Nodes

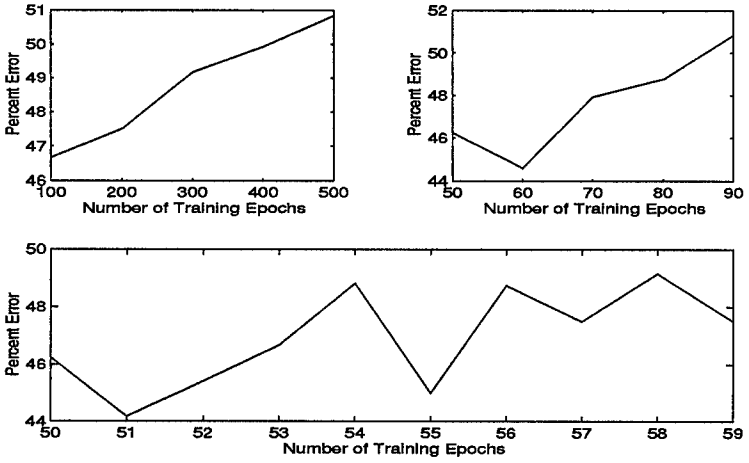


Figure 31. Testing Error History Plot of the 200 Point Test Set to Determine the Number of Training Epochs

In order to determine how training is affected by the initial conditions of the neural network, the initial weights are varied to observe the effect upon the classification of the test sets. The initial weights are set by picking the random seed used by LNKnet. Five different seeds are used, in order to ensure that a good range of initial weights is used, which causes a variance in the results.

Data Set	Seed 0	Seed 1	Seed 2	Seed 3	Seed 4
	(%)	(%)	(%)	(%)	(%)
860	60	50	55	50	40
880	60	50	45	55	60
900	50	60	40	45	55
920	80	65	75	65	70
940	50	50	70	55	55
960	80	55	75	65	75
1000	65	75	60	60	75
1020	65	65	70	50	55
1040	30	30	50	35	35
1060	50	40	50	50	45
AVG	59.00	54.00	59.00	53.00	56.50

Table 14. S and P DVS and Neural Network: MLP: 7,4,2

The network is trained with 850 data vectors. The results range from 53 percent correct to 59 percent correct, with all of the sets' accuracies above the initial accuracy obtained by the DVS algorithm. Either random seed 0 or 2, which determine 59 percent test accuracy, may be chosen as the best implementation, dependent on whether a few major changes of the individual 20 point test set prediction accuracies are preferred over small changes of several individual test set prediction accuracies.

For the INFFC data set, the number of inputs is 3. The DVS algorithm does not determine an embedding dimension for this set, but rather shows that the data is stochastic. The embedding dimension is thus chosen to minimize the run time of the neural networks. The number of outputs is 3, enough to classify the data as up, down, or hold. The number of middle nodes ranges from the maximum number, 200, down. Using the 200 point test set, the test errors are taken after 100 epochs.

The testing error does not change for any number of hidden nodes. The network classifies all of the test data vectors as holds.

In order to see if the neural network will learn if trained longer, the number of training epochs is varied from 500 epochs down. Again using the 200 point test set, the test errors do not change for any number of training epochs. All of the points are still classified as holds. The number of training epochs and the architecture could not be set to induce the neural network to learn the test data properly. Indeed, the neural network would not even learn the training set, moving around from value to value randomly.

4.5 Nearest Neighbor Neural Adaptation

Using the nearest neighbors as a neural training set is an attempt to create a local prediction method [3]. These procedures are designed to determine if a local set of data, i.e. the nearest neighbors, is able to train a neural network so that its performance is at the same, or better, level of prediction accuracy as the DVS prediction set. The nearest neighbors are those vectors which most closely mirror the trajectory of the attractor when the test vector occurs.

The first set of experiments is a top down approach to determine the best architecture and training size of the multilayer perceptron and radial basis function neural networks with respect to the large training set. The MLP and RBF architectures which provide the best testing accuracy for both data sets are chosen. The second set of experiments is designed to create a more generalized neural network since a higher level of generalization is normally an indicator of a more successful network for use in the world where the data was taken.

For the S and P data set, the architecture for the optimized neural network is determined to be 7 inputs, 3 middle nodes, and 2 outputs for the multilayer perceptron and 7 inputs, 2 outputs, and 16 cluster centers for the radial basis function. The multilayer perceptron achieves an average test accuracy of 56.50 percent, five

Data Set	Prediction Accuracy
	(%)
860	45
880	60
900	50
920	55
940	55
960	70
1000	55
1020	60
1040	60
1060	55
AVG	56.50

Table 15. S and P DVS Nearest Neighbor MLP: 7,3,2

Data Set	Prediction Accuracy
	(%)
860	45
880	25
900	55
920	45
940	40
960	55
1000	50
1020	55
1040	40
1060	35
AVG	44.00

Table 16. S and P DVS Nearest Neighbor RBF: 7,2, Centers: 16

percent higher than the DVS prediction of 51.50 percent. Notice that the test sets with the highest and lowest accuracy rates are not consistent across the two methods. The MLP's most accurate test set is 960 and the least accurate is 860. The DVS's most accurate test set is 860 and the least accurate is 1000. This provides some incentive to create a fusion method for the algorithms, in order to exploit the disparate testing accuracies. The radial basis function does not perform as well as the DVS algorithm, with an accuracy of 44 percent. The individual test sets do not do very well, with 6 of them below 50 percent accurate.

Data Set	Prediction Accuracy (%)
860	40
880	65
900	55
920	65
940	55
960	60
1000	45
1020	55
1040	50
1060	40
AVG	53.00

Table 17. S and P DVS Nearest Neighbor MLP: 7,1,2

The architecture of the generalized neural network is determined to be 7 inputs, 1 middle node, and 2 outputs for the multilayer perceptron and 7 inputs, 2 outputs, and 8 cluster centers for the radial basis function. The multilayer perceptron achieves an average test set accuracy of 53 percent, 1.5 percent higher than that achieved by the DVS algorithm. The radial basis function again does not achieve the DVS accuracy, with an average test set accuracy of 45.50 percent. Again, the test sets are disparate across the methods, warranting the use of a fusion algorithm.

For the INFFC data set, the architecture for the optimized neural network is determined to be 3 inputs, 2 middle node, and 3 outputs for the multilayer perceptron

Data Set	Prediction Accuracy
	(%)
860	40
880	50
900	45
920	45
940	45
960	55
1000	45
1020	60
1040	35
1060	55
AVG	45.50

Table 18. S and P DVS Nearest Neighbor RBF: 7,2, Centers: 8

Data Set	Prediction Accuracy
	(%)
25000	15
25100	30
25500	30
25600	15
26000	20
26100	30
26500	35
26600	30
27000	30
27100	15
AVG	25.00

Table 19. INFFC DVS Nearest Neighbor MLP: 3,1,3

Data Set	Prediction Accuracy (%)
25000	10
25100	50
25500	40
25600	45
26000	20
26100	40
26500	55
26600	45
27000	40
27100	40
AVG	38.50

Table 20. INFFC DVS Nearest Neighbor RBF: 3,3 Centers: 16

and 3 inputs, 3 outputs, and 16 centers for the radial basis function. The neural network architecture which is designed to generalize the data is not implemented for this test set since it did not prove very successful with the S and P data.

The multilayer perceptron achieves an average test set accuracy of 25 percent, much lower than the DVS prediction accuracy of 35 percent. The radial basis function performs better than the MLP, with an average prediction accuracy of 38.50 percent. The test sets with the highest and lowest accuracy rates again are not consistent across the two methods. The RBF's most accurate test set is 26500 and the least accurate is 25000. The DVS's most accurate test set is 25100 and the least accurate is 25500. This disparity again provides incentive to create a fusion method to tie the algorithms together in an attempt to raise the prediction accuracy.

4.6 Bayes' Bounding Error Probability

The Bayes Error Probability is used to determine a range of probable error for a given data set based upon the set of features. These calculations will enable the determination of the upper bound for the prediction accuracies for the given data set. The Bayes Error Probability is found for both the k-nearest neighbors and

Parzen Windows algorithm. This algorithm tells if the prediction algorithms are giving accuracies which approximate the Bayes Probability.

Examining the S and P data, the undetrended and First Difference errors are bounded. The embedding dimension is 7, as determined by the DVS algorithm. This number is used to determine the number of features. Using 2 different leave-one-out methods, the optimal k for the k-nearest neighbor method is determined for both leave-one-out methods. The optimal h for the Parzen windows method is also found for both leave-one-out curves.

For the raw S and P data, the error boundaries converge upon a lower bound of 45 percent and an upper bound of 50 percent for both the k-nearest neighbor and Parzen window methods. For this data set, only one leave-one-out option gives an error estimate. The other option does not present results. The optimal k value is determined to be 2, while the optimal h value is determined to be 0.8.

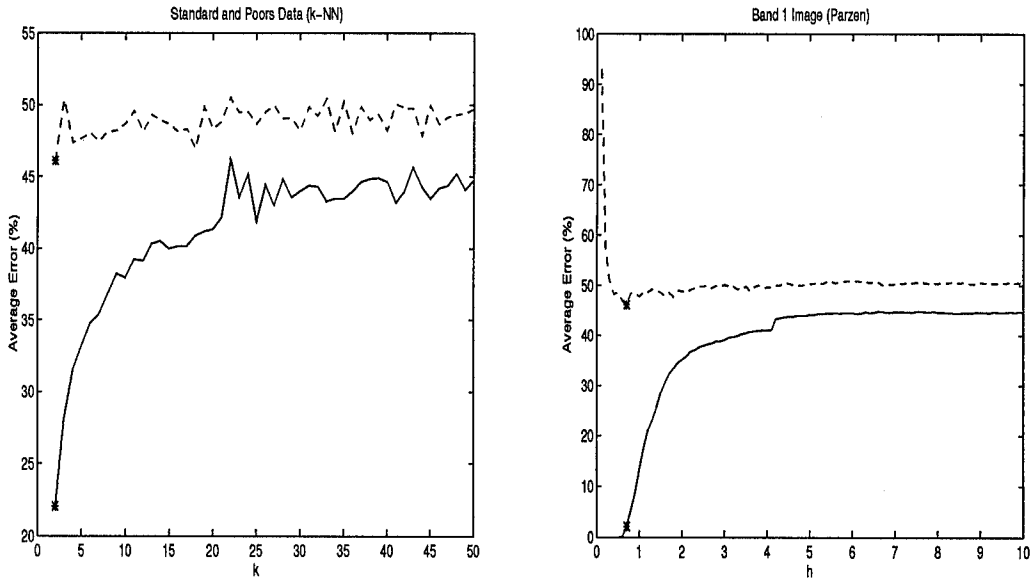


Figure 32. Bayes Bounding of Raw SandP data, m=7

For the First Differences S and P data, the error boundaries are different for the two methods. Both leave-one-out options give error estimates for this data set.

The k-nearest neighbor error boundaries converge to an upper bound of 28 percent with a lower bound of 26 percent. The optimal k, determined from the dotted line, is 43. The Parzen Window error boundaries converge to an upper bound of 28 percent with a lower bound of 25 percent. The optimal h value is 2.2.

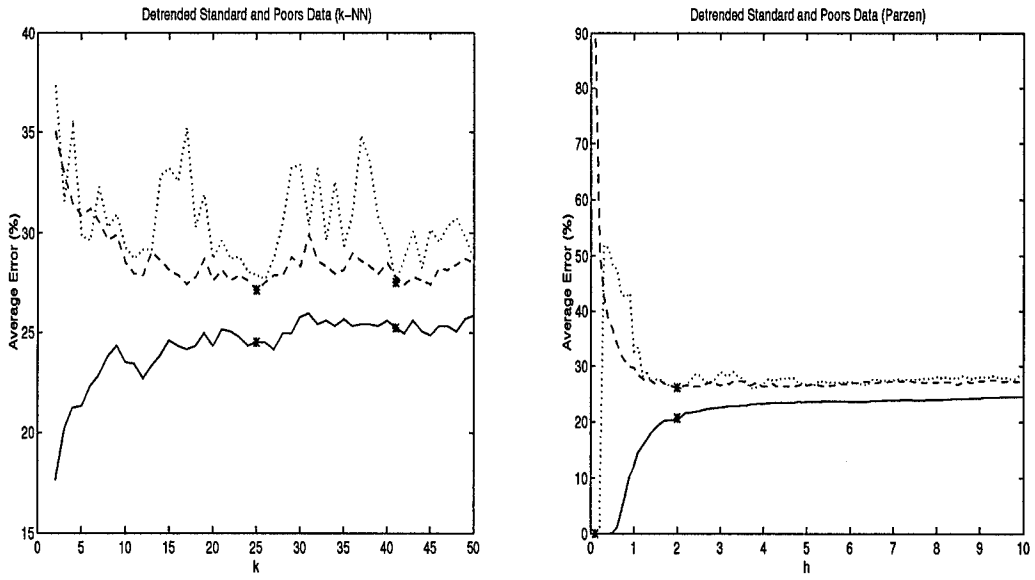


Figure 33. Bayes Bounding of First Differences SandP data, m=7

For the INFFC data, the code is run three times. Since this data is made up of three classes, the Martin Code must be run for all combinations of 2 classes that exist. The embedding dimension is 3, as discussed in Section 4.4. This number determines the number of features. The three runs, for classes 01, classes 02, and classes 12, are combined in order to determine the optimal k for the k-nearest neighbor method and the optimal h for the Parzen windows method.

For the INFFC data, the k-nearest neighbor plots all converge upon an upper bound of 50 percent error. The lower bound for the class 01 set converges upon 47 percent error. The lower bound for the class 02 set converges upon 10 percent error. The lower bound for the class 12 set converges upon 18 percent error. For all of the class sets, the error starts at zero, and rises as k increases. This seems to indicate that the more information which is presented, the more confused the classes

become. The first 10 values for k are optimal for all of the three class sets. For the Parzen windows method, the upper bounds for the class 01 set converges upon 48 percent error, the class 02 set converges upon 40 percent error, and the class 12 set converges upon 43 percent error. The lower bounds for the class 01 set converges upon 47 percent error, the class 02 set converges upon 38 percent error, and the class 12 set converges upon 42 percent error. The optimal h is 2.2 for class 01, 5.3 for class 02, and 7.1 for class 12.

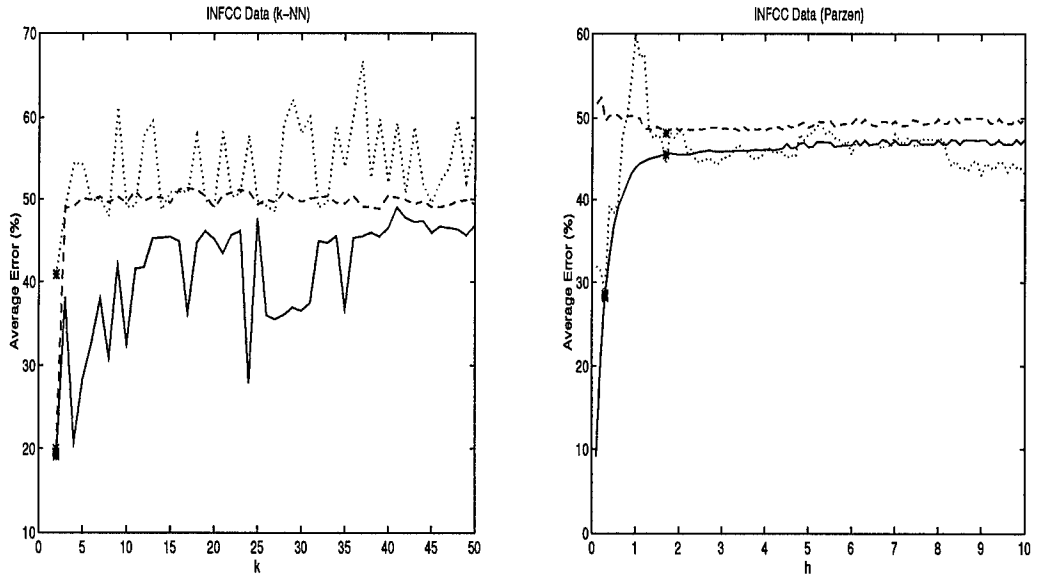


Figure 34. Bayes Bounding of Raw INFFC data, m=3, classes 0,1

4.7 Bayesian Classifiers

Having used the Bayesian Error Probabilities to determine the optimal k, a method designed to use this k value for classification is the next step. The k-Nearest Neighbor algorithm is thus chosen to implement the results of the Bayes Bounding in order to determine the validity of Martin's Code for this problem, as well as try to improve upon the accuracy obtained by the DVS algorithm. The k-Nearest Neighbors algorithm is implemented using the LNKnet software package.

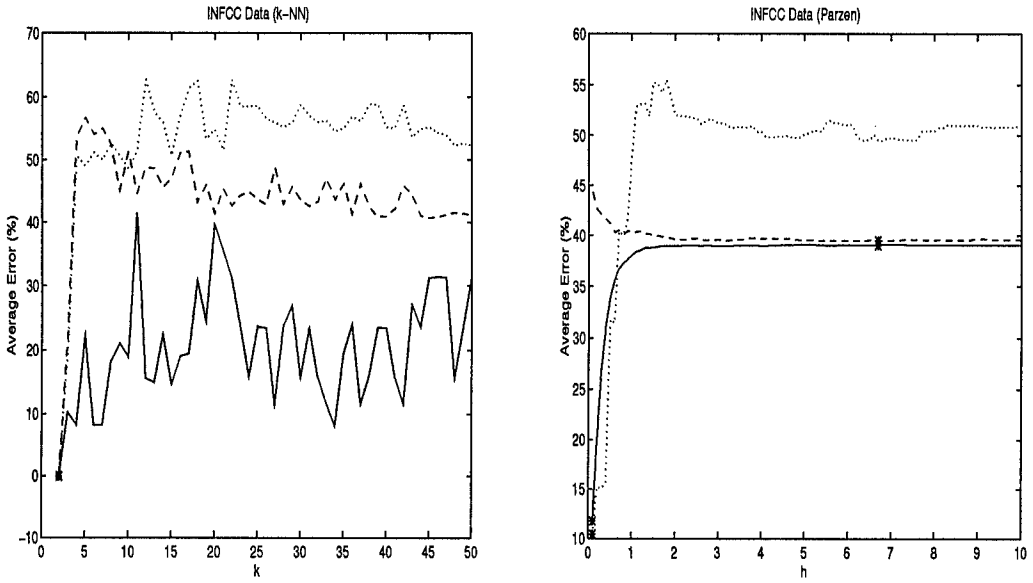


Figure 35. Bayes Bounding of Raw INFFC data, m=3, classes 0,2

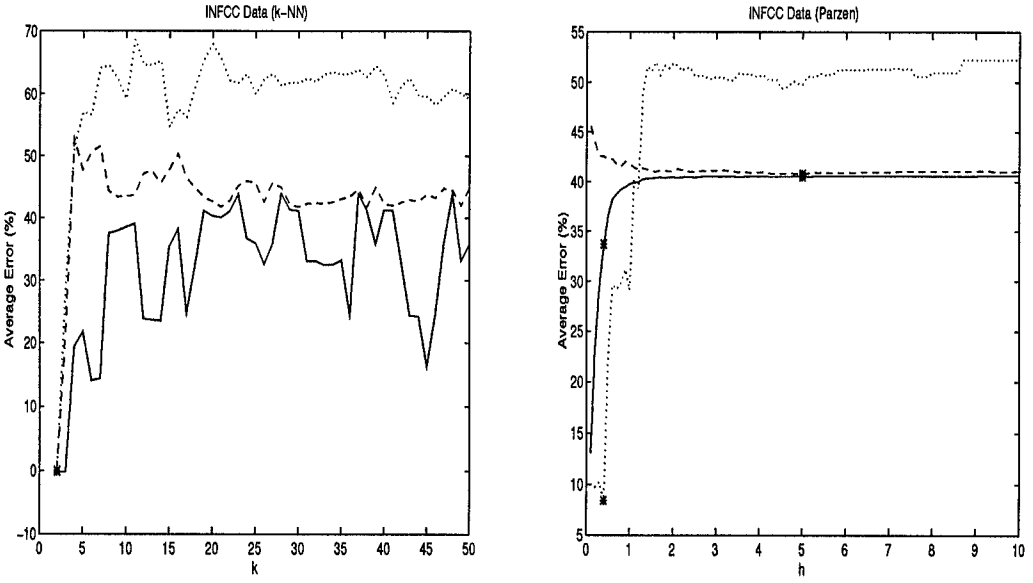


Figure 36. Bayes Bounding of Raw INFFC data, m=3, classes 1,2

Data Set	k=16	k=25	k=42
	(%)	(%)	(%)
860	55	55	55
880	55	50	40
900	55	45	50
920	50	60	65
940	50	65	30
960	60	70	55
1000	70	70	70
1020	65	60	55
1040	35	35	40
1060	55	55	50
AVG	55.00	56.50	51.00

Table 21. S and P Bayesian Classifier: k-NN with Varying k

For the First Differences S and P data, several k values are examined. The first k value is 16, which is the k found using the DVS algorithm. The second k value is 25 and the third is 42, the optimal k values found using the two options of the Bayesian Error Probability Estimation. The individual test set accuracies are calculated, as well as the average test set accuracy. The second k value, k=25, provides the best average test set accuracy, with an accuracy of 56.50 percent, which is below the Bayes bounding error range determined by the Martin Code.

Examining the individual test set accuracies, and comparing them to the DVS test accuracies, the disparity between the sets is evident. For the DVS set, the most accurate test set is 860 and the least accurate test set is 1000. For the k=25 Bayesian classifier, the most accurate test sets are 960 and 1000 while the least accurate is 1040. This disparity between the sets provides great incentive to create some type of fusion algorithm.

For the INFFC data, several k values are also examined. The first k value is 5, determined by the Bayes Error Probability curves. The second k value is 19 and the third k is 100, since the data is determined to be stochastic by the DVS algorithm.

Data Set	k=5	k=19	k=100
	(%)	(%)	(%)
25000	20	10	15
25100	50	25	30
25500	45	30	20
25600	30	35	40
26000	30	30	35
26100	25	50	30
26500	40	25	20
26600	40	35	40
27000	50	35	45
27100	40	45	40
AVG	37.00	32.00	31.50

Table 22. INFFC Bayesian Classifier: k-NN with Varying k

The individual test set accuracies are calculated, as well as the average test set accuracies. The first k value, k=5, provides the best average test set accuracy, with an accuracy of 37.00 percent.

Examining the individual test set accuracies, and comparing them to the DVS test accuracies, the disparity between the sets is again evident. For the DVS set, the most accurate test set is 25100 and the least accurate test set is 25500. For the k=5 Bayesian classifier, the most accurate test sets are 25100 and 27000 while the least accurate test set is 25000. This disparity provides some incentive to creating fusion algorithms.

4.8 Fusion of the Algorithms

A fusion method tries to combine several algorithms in order to increase prediction accuracy. Trying to fuse the methods together is only effective if the methods' errors are complimentary. Errors are complimentary if the occur at different times in the test sets. A successful fusion method chooses the algorithm with the best prediction accuracy for the given test set.

The first fusion method is called the majority rule without rejection. This method incorporates an odd number of algorithms and examines the classification for each point. The class which receives the majority vote, i.e. is chosen by the most algorithms, is the winner. This process is repeated for all of the data points in the test set. No rejection is implemented since the DVS and Spectral Estimation algorithm is not considered in this method.

The second fusion method is the average fusion method. this method incorporates a various number of algorithms, from two to five algorithms at a time. $P_A(i)$ is found using the algorithms' postprobabilities. The class i with the largest $P_A(i)$ is chosen as the winner. This method is used for all the data points in the set.

The third fusion method is the weighted probability fusion method. The method uses the best combination of the algorithms as found by the average fusion method. The v algorithms are weighted by their individual successes. Once the initial weights are derived, the fusion method is run. The prediction accuracy is calculated, and the weights are changed until a threshold probability is found and the weights are set.

Data Set	Prediction Accuracy
	(%)
860	65.00
880	60.00
900	50.00
920	70.00
940	65.00
960	80.00
1000	50.00
1020	60.00
1040	40.00
1060	55.00
AVG	59.5

Table 23. First Differences S and P Fusion Method # 1: Majority Rule

For the First Differences S and P data set, the majority rule fusion method uses the neural network, nearest neighbor neural adaptation, and the k-Nearest Neighbor algorithms to classify the data as up and down. This fusion method works well, obtaining an average prediction accuracy of 59.50 percent. No individual test set accuracies for the fusion methods drop below 40 percent, which is higher than the least accurate test set for any of the individual algorithms.

Algorithms	Average Prediction Accuracy (%)
DVS, k-NN, net1, nn	48.5
DVS, k-NN, net2, nn	49.00
DVS, k-NN, net1, net2, nn	50.00
DVS and k-NN	53.50
DVS and nn	49.00
DVS and net1	58.00
DVS and net2	52.50
DVS, net1, net2	57.00
k-NN and net1	56.50
nn and net2	49.00
net1 and net2	57.00

Table 24. First Differences S and P Fusion Method # 2: Average Probability Fusion

The average fusion method uses the postprobabilities for the DVS, the neural network for two different random seeds, the nearest neighbor neural adaptor, and the k-nearest neighbor algorithms. As seen in Table 24, ten different combinations of the algorithm are examined. Judging from the average prediction accuracy, the best combination of the algorithms is the DVS and the first nearest neighbor algorithms. This combination achieves an average prediction accuracy of 58 percent, below that of the majority rule algorithm.

The weighted probability fusion method uses the DVS and first neural network algorithm combination to try to improve upon the average fusion method. The weights are initially set by the algorithms' prediction accuracies. After several calculations, the weights are set at $W_1 = 0.1$ and $W_2 = 0.25$. The average prediction

Algorithm	DVS and net1
Data Set	Prediction Accuracy
	(%)
860	60.00
880	55.00
900	50.00
920	80.00
940	55.00
960	75.00
1000	70.00
1020	50.00
1040	45.00
1060	45.00
AVG	58.5

Table 25. First Differences S and P Fusion Method # 3: Weighted Probability Fusion

accuracy, found in Table 25, is 58.50 percent, above the average fusion method, but below the majority rule method.

For the raw INFFC data set, the majority rule fusion method uses the neural network, the nearest neighbor neural adaptor, and the k-nearest neighbor algorithms to classify the data as up,down, and hold. This fusion method does not work very well, obtaining an average prediction accuracy of 26.50 percent, well below the probability of chance.

The average fusion method uses the probabilities for the DVS, k-nearest neighbor, and two nearest neighbor neural adaption algorithms. The first nearest neighbor set, nn, is the raw postprobabilities of the algorithm. This set contains NaNs, which stands for either infinity or minus infinity. The second nearest neighbor set is nn1, which is the postprobabilities with the NaNs replaced with zeros, which allows these points to have no bearing upon the outcome of the fusion. As seen in Table 27, nine combinations of the algorithms are examined. The best combination of the

Data Set	Prediction Accuracy
	(%)
25000	5.00
25100	55.00
25500	30.00
25600	30.00
26000	25.00
26100	20.00
26500	30.00
26600	35.00
27000	35.00
27100	30.00
AVG	26.50

Table 26. Raw INFFC Fusion Method # 1: Majority Rule

Algorithms	Average Prediction Accuracy
	(%)
DVS, k-NN, nn	38.50
DVS, k-NN, nn1	38.00
DVS, k-NN, nn, nn1	39.00
DVS and k-NN	38.50
DVS and nn	37.50
DVS and nn1	35.50
k-NN and nn	38.50
k-NN and nn1	36.00
nn and nn1	38.50

Table 27. Raw INFFC Fusion Method # 2: Average Probability Fusion

algorithms is DVS, k-nearest neighbors, and both nearest neighbor neural adaption sets. This combination achieves an average prediction accuracy of 39 percent.

Algorithm	DVS, k-NN, nn, nn1
Data Set	Prediction Accuracy
	(%)
25000	30.00
25100	45.00
25500	40.00
25600	45.00
26000	25.00
26100	35.00
26500	60.00
26600	40.00
27000	30.00
27100	45.00
AVG	39.50

Table 28. Raw INFFC Fusion Method # 3: Weighted Probability Fusion

The weighted probability fusion method uses the best algorithm combination, as determined by the average prediction method. The weights are initially set by the algorithms' individual prediction accuracies. After setting the initial weights at $W_1 = 0.1$ for the DVS data, $W_2 = 0.2$ for nn and nn1 data, and $W_3 = 0.4$ for the k-NN data, the method is run. After many calculations, the weights are set at W_1 = -0.1, $W_2 = 0.1$, and $W_3 = 0.4$. The average prediction accuracy, from Table 28, is 39.5 percent, the best achieved by any of the fusion methods for this data set.

4.9 Statistical Significance of the Algorithms

The Statistical Significance of the various experimental algorithms with respect to the DVS algorithms is also calculated.

Algorithm	t-score	Confidence Interval
	Magnitude	Magnitude
DVS	-	7.5494
Spectral	0.7682	10.1970
k-NN	0.8733	7.9170
Neural	1.2247	4.7836
NN neural	1.1028	10.7693
Fuse #3	1.1290	8.1645
Fuse #1	1.7143	8.9294

Table 29. t-Scoring and Confidence Intervals for the S and P data

Algorithm	t-score	Confidence Interval
	Magnitude	Magnitude
DVS	-	6.9105
Spectral	0.1493	11.7172
k-NN	0.6429	7.3877
NN neural	0.7297	9.69263
Fuse #1	0.9493	8.9929
Fuse #3	1.0287	7.2419

Table 30. t-Scoring and Confidence Intervals for the INFFC data

4.10 Summary

This Chapter examines and discusses the results of the individual classification algorithms and the fusion methods described in Chapter III. The following tables present the significant results from the experiments conducted for this thesis, including the error bounds determined from the Bayes Error Probabilities.

Algorithm	Prediction Accuracy
	(%)
Neural Network, seeds 0,2	59.00
Nearest Neighbor Neural, MLP, k=16	56.50
k-NN, k=25	56.50
DVS and Phase, k=16	54.67
DVS, k=16	51.50

Table 31. Algorithms using S and P: Ranking from Best to Worst

Algorithm	Prediction Accuracy
	(%)
Nearest Neighbor Neural, RBF, k=19	38.50
k-NN, k=5	37.00
DVS, k=5	34.00
DVS and Phase, k=5	33.17

Table 32. Algorithms using INFFC: Ranking from Best to Worst

Algorithm	Prediction Accuracy
	(%)
Majority Rule	59.50
Weighted Probability	58.50
Average Probability	58.00

Table 33. Fusion Methods using S and P: Ranking from Best to Worst

Error bounds are determined from the Bayes Error Probability.

Algorithm	Prediction Accuracy
	(%)
Weighted Probability	39.50
Average Probability	39.00
Majority Rule	26.50

Table 34. Fusion Methods using INFFC: Ranking from Best to Worst

S and P error: (k) (01) upper: 35% lower: 25%
(h) (01) upper: 28% lower: 20%

INFFC error: (k) (01) upper: 60% lower: 48%
(k) (02) upper: 60% lower: 30%
(k) (12) upper: 60% lower: 40%

(h) (01) upper: 50% lower: 47%
(h) (02) upper: 50% lower: 39%
(h) (12) upper: 50% lower: 41%

Conclusions concerning the validity of the classification algorithms and the fusion methods, as well as some suggestions concerning additional work in this research area, are presented in Chapter V.

V. Conclusions and Recommendations

5.1 Introduction

Chapter V examines the results of the algorithms for both data sets with respect to the overall emphasis of this thesis. The success rate of the individual algorithms is examined, as well as the success of the various fusion methods. The ability to train a neural network upon a limited data set is also examined, as well as the determination of the error bounds based upon the Bayes Error rates.

This chapter also presents some recommendations for further work to be done in this field of study. The first examines the use of the detrended INFFC data set to check for improvement over the raw data, while the second examines the addition of phase to the training vector as an additional feature. The third recommendation examines the development of a nearest neighbor confidence measure, while the fourth examines pruning the nearest neighbors for the nearest neighbor neural adaptor. The final recommendation the Mahalanobis distance metric k-Nearest Neighbor algorithm.

5.2 Conclusions

The focus of this thesis is to increase the prediction accuracy of current time series prediction algorithms. This can be readily measured by comparing the algorithms' prediction accuracies to that of the DVS algorithm. For the Sand P data set, all four of the experimental algorithms tested surpassed the prediction accuracy of the DVS algorithm. The specific parameters needed for each algorithm are described in Chapter IV. For the INFFC data set, only two of the experimental algorithms tested surpassed the prediction accuracy of the DVS algorithm. The specific parameters needed for each algorithm are again described in Chapter IV.

The fusion algorithms are attempts to combine the algorithms in order to capitalize upon their individual successes. The fusion methodologies form two categories,

the majority rule voting algorithm and the probabilistic classification algorithms. For the S and P data set, the majority rule algorithm provides the best prediction accuracy, which also surpasses the prediction accuracies of all the individual algorithms. For the INFFC data set, the weighted probability algorithm provides the best prediction accuracy, which also surpasses the prediction accuracies of the individual algorithms. In both cases, the fusion algorithms surpass the prediction accuracies achieved by any one prediction algorithm.

Martin Casdagli raised several interesting questions in his article written for the Santa Fe Competition. One of these is the ability to use a limited local training set to predict the next point of a time series [3]. This question led to the neural network algorithm, trained with the nearest neighbors, used in this thesis. This algorithm uses the nearest neighbors for the last vector of the fitting set to predict the next point. Both the MLP and RBF architectures are used to determine which algorithm best suits the data set. For the S and P data set, the MLP architecture is successfully trained with the nearest neighbor set consisting of the sixteen training vectors. The prediction accuracy of this algorithm surpasses that of the DVS algorithm, ranking second out of the five tested algorithms. For the INFFC data set, the RBF architecture is more successful in training with the nearest neighbor set consisting of nineteen training vectors. The prediction accuracy of this algorithm surpasses the DVS accuracy, and is the best algorithm out of the four successfully tested. These results indicate that locally training an algorithm can achieve the same, or better, results than using the entire training set.

The Bayes Error Probabilities are found in order to determine probabilistic error bounds for each of the data sets. These bounds should be able to determine the success of the prediction algorithms. The success is measured by the nearness of the the algorithms' prediction accuracies to the bounds found by the Martin Code. For the S and P data set, none of the algorithms' prediction accuracies approach the error bounds found by the Bayes Error Probability. This shows that there is room for

improvement for the prediction accuracy of the algorithm to reach the Bayes Error Probability bounds. For the INFFC data set, the algorithms' prediction accuracies seem to approach the error bounds. This is hard to detect since an exact error bound for the three class problem has not been determined. The algorithms are at least closer to the bounds evident from the two class component bounds. The Bayes Probability bound shows that some improvement can be made in both data sets.

Examining the t-scores determines the statistical significance of the various experimental algorithms with respect to the DVS algorithm. For the S and P algorithms, none of the algorithms have t-score magnitudes which surpass 1.833. This shows that all of the algorithms are statistically similar with a 95 percent confidence interval. Notice that the t-score magnitude gets bigger as the algorithms' average prediction accuracy increases. The fusion method with the best average prediction accuracy amongst all the algorithms is also the one with the largest t-score. The more the experimental algorithm differs from the DVS algorithm, the better the prediction accuracy is. For the INFFC algorithms, none of the algorithms have t-score magnitudes which surpass 1.833. This shows that all of the algorithms are statistically similar. The t-scores again increase as the algorithms' average prediction accuracy increases. The more the experimental algorithm differs from the DVS algorithm, the better the prediction accuracy is. This leads to the conclusion that, for these experimental algorithms, the more statistically significant the algorithm, the better the prediction accuracy it attains.

5.3 Recommendations

There are several questions which arise from the conclusion of this thesis. Finding the answers to a particular problem inevitably leads to more questions concerning the actual answers to the problem. This section examines some of the work which can be done to answer the new questions which have recently surfaced.

The detrended INFFC data set needs to be run through the algorithms to see if it performs better than the raw data. All evidence suggests that detrending the data aids in prediction, as discussed in Section 3.2. This set has not been done since the DVS algorithm implemented by Stright can not handle the large amount of data, and the DVS algorithm implemented by Stewart can not be properly run with strings of zero in the data, which the detrended set contains. Running this set through the algorithms should significantly improve the prediction accuracy for this set.

A new way of using the phase information to aid in prediction needs to be examined. Instead of using the phase to reject points as in the current algorithm, the phase should have some value as a feature. One modification which can easily be made is to add the phase component to the current neural network feature vector as a new feature. This makes the features contained in the training vector the n time series points which make up the n-tuple and the phase component. This will hopefully be a significant feature and increase the prediction accuracy.

A new use for the nearest neighbor probabilities has been suggested . the relative probability of what direction the nearest neighbors are moving in can be determined from the number of neighbors which are moving in the same direction. For example, if six neighbors are moving upward and four neighbors are moving downward, the supposition is that there is a sixty percent probability that the next point is moving along the upward trajectory and a forty percent probability that it is moving downward. These probabilities can be used as a confidence measure for the prediction. If the prediction is up, and there is more than a fifty percent probability that the next point will move up, then the prediction is said to be confident. If the prediction is down, however, and there is more than a fifty percent probability that the next point is up, then the prediction is no longer confident. This method can be used in conjunction with any algorithm which incorporates the use of the nearest neighbors.

If the training set can be reduced down to the nearest neighbors, can it be reduced any further? This effort to further prune the nearest neighbors can be done if several training vectors present nearly identical information to the algorithm. In order to determine if a training vector is extraneous, a leave-one-out methodology must be employed upon the training set. This allows the algorithm's output to analyzed with each training vector removed from the nearest neighbor training set. Once a determination is made concerning the applicability of each training vector to the successful training of the algorithm, those vectors deemed unnecessary can then be removed. The remaining vectors make up the pruned nearest neighbors training set.

Another recommendation is to create a Mahalanobis distance k-Nearest Neighbor algorithm. The Bayes Error Bounding is based upon code which uses the Mahalanobis distance as the metric. All of the algorithms used in this thesis use the Euclidean distance metric. Since the algorithms' performances are based upon the Bayes Error Bound, the difference in the distance metric is a factor which must be considered. It must be determined if algorithms using the Mahalanobis distance metric perform better than their Euclidean counterparts. Creating the Mahalanobis k-Nearest Neighbors algorithm is the first step in this endeavor. This particular algorithm is chosen since the Bayes Error Bounding finds an optimal k value using the Mahalanobis metric. Along the same lines of thought, the creation of a Parzen Windows classifier will provide further evidence for the adoption of the Bayes Error Probability as a reliable method of the determining the data's actual error bounds.

The final recommendation is to combine the two class probabilities of the multiclass problem in order to derive a single upper and lower bound error set with respect to all of the classes. This problem is a statistical one dealing with the combination of several probability curves determined using PDF estimators. Looking at the individual two class probabilities does not convey enough information concerning the probability set for the multiclass space to reliably make a bounds

determination. Only if this problem is solved can multiclass prediction problems be better understood through the analysis of the algorithms' success with respect to the error bounds.

5.4 Summary

This Chapter has presented conclusions concerning time series prediction based upon the results of the experiments run for this thesis. These conclusions are followed by the recommendations on how to continue the work presented here for further exploration and possible improvement of current algorithms designed to perform time series prediction.

Appendix A. LNKnet Summary

LNKnet is a software package which impliments neural and statistical pattern recognition algorithms. It runs under the UNIX system in SUN Openwindows, with the keyboard and mouse as the input devices. Over 20 static pattern classification, clustering, and feature selection algorithms are presented with graphical outputs as wellas confusion matrices and error history lists. Classifiers have been successfully trained using more than 1000 features and 100,000 training patterns. To start LNKnet, type 'LNKnet' at the command line. This will bring up the main LNKnet window. It is advisable that LNKnet is started in the home directory, so that any other directory may be accessed by it.

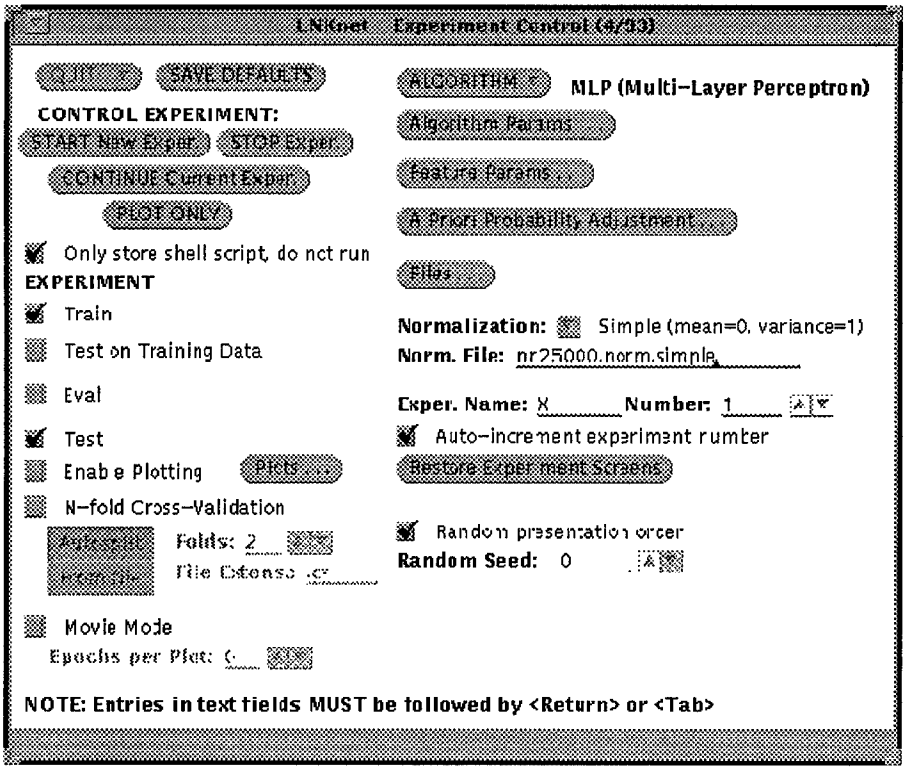


Figure 37. The Main LNKnet Window [16]

There are many algorithms presented in LNKnet, for supervised and unsupervised training. Neural network algorithms, conventional pattern classification

algorithms, and feature selection algorithms are all present, with several examples of each algorithm type. The algorithms used in this thesis are the multilayer perceptron neural network, the radial basis function neural network, and the k-nearest neighbor algorithm using k-means clustering. This section presents guidelines to run any of the algorithms.

For use with any of the algorithms, the data files need to be formatted for use with LNKnet. The data base is usually split into three files: the training file, the evaluation file, and the testing file. The training file is used to train the classifier. The evaluation file is used to evaluate the classifier in order to determine the classifier size and tune classifier parameters. The testing file is used to determine the final error rate. For smaller data sets, the evaluation file may be eliminated.

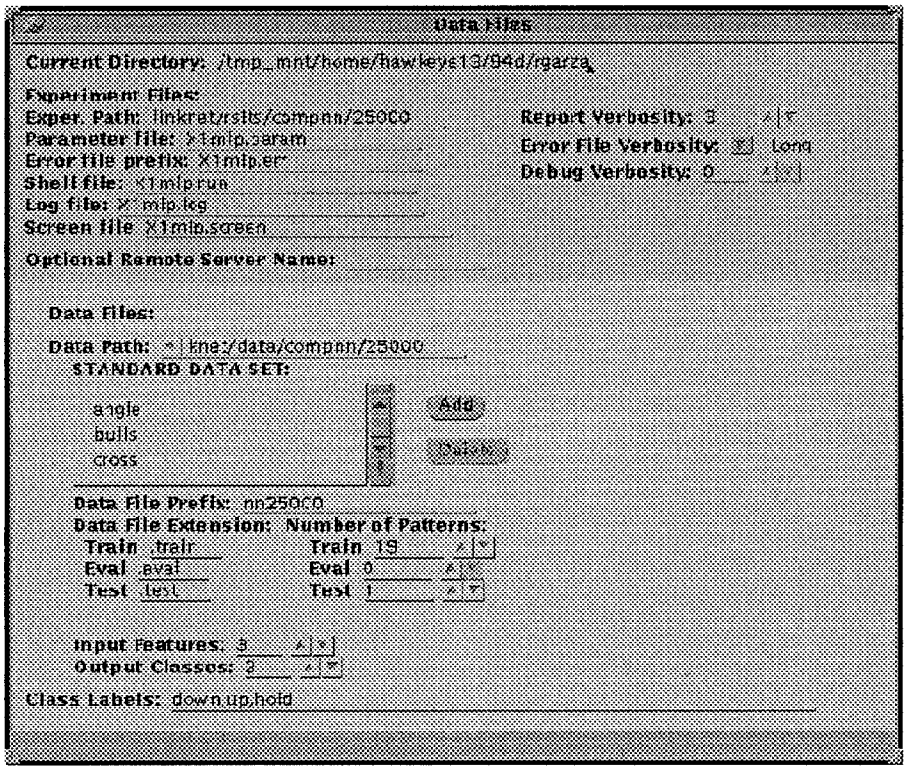


Figure 38. The Files Window [16]

The format of these files is identical. All files are ASCII, with one pattern per line. The first number of each pattern is an integer determining the class, from zero to the number of classes minus one. The remaining numbers are floating point values of the features of the pattern, each seperated by a space or a tab. The last line must have a carriage return at the end.

The *files* window is used to select the data base. The names of data files are generated by adding extensions to the data base name. The default extensions, used with the sandp data base, are sandp.train, sandp.eval, and sandp.test. Each of these data files must have a corresponding default file. These default files contain a list of flags and their values. The defaults for a particular data file are named <datafiles>.defaults. The file must be set up in the following fashion:

describe -ninputs 7 -noutputs 2 npatterns 20 -labels down,up

Every training, evaluation, and testing files need one.

The LNKnet algorithms generate several files. The files which are pertinent for evaluation of the algorithm are the log and error files. The log file contains information saved from the run in the .log file, with the amount of information dependent upon the **verbosity** choice. The higher the verbosity level, the more information which is retained. In most cases, the verbosity level should be set to 3, the highest level, since a loss of information concerning the running of the algorithm may be damaging to future research.

When a classifier is used, the classification results for each pattern can be stored in an error file. The **Error File Verbosity** is chosen to set the amount of information contained by the error file. If the error file verbosity is set to **None**, no error file is written at all. If set to **Short**, the error file contains the following information:

pattern # - correct class - classifier's class - classification error - cost

If set to **Long**, the error file contains the information contained in the IShort file plus the following information:

normalized input parameters (nodes) - classifier output parameters (nodes)

The file name extension for these files is .err.

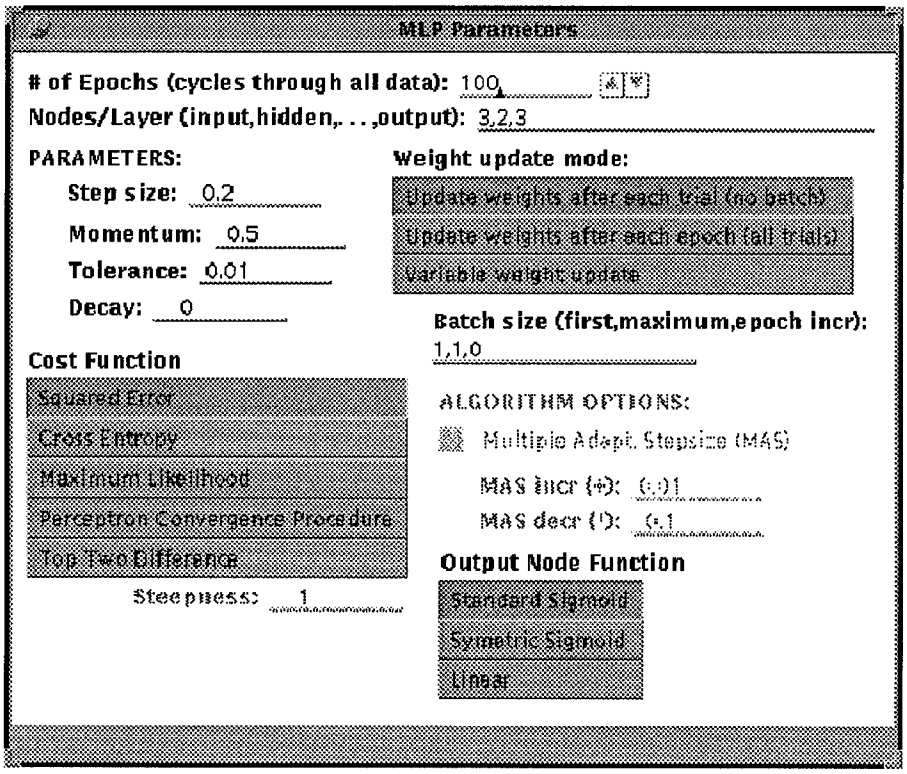


Figure 39. The Algorithm Parameters Window for the MLP [16]

Once the data base files are correctly set up, the algorithm is almost ready to run. The **Algorithm** menu on the *Main LNKnet* window allows the user to chose an algorithm. Pressing the **Algorithm Parameters** button on the *Main* window allows the user to completely set up the algorithm. Pressing the **Files** button brings up the *Files* window. The correct **Data Path** is set by the user, including all of the folders leading up to the user's home directory. The **Experiment Path** is set to write to the folder where the user wants to write the output, starting from the

current directory where LNKnet is being run. With the files and paths set up, the algorithm is now ready to run.

To actually run the experiment, use the buttons under the **Experiment** heading on the *Main* window. Press the **Train** button and the **Test** button. If the experiment is to be run immediately, press the **Start New Exper.** button. If the experiment is to be run later, or in the background, press the **Store Shell Script** button before pressing the **Start** button. If the data trains, but does not test, rerun the experiment with only the **Train** button pressed. Press the **Train** button again to remove the check mark and press the **Test** button. Press the **Continue Experiment** button to run the test files. The experiment can only be continued if the shell script has already been run once.

If the experiment is saved to a shell script, it may be run in the background. The shell script file is named <filename>.run. To run this shell script in a command window, change directories until the directory with the shell sript in it is chosen. Type '<filename>.run' and press return. The shell script now runs as normal. To run the shell script in the background, change directories in the command tool to the one which contains the experimental files. Type 'NICE <filename>.run &' and press return. The files may be run without the NICE mode designation, but this is not advisable unless no one is scheduled to be using the particular SUN station for a lengthy amount of time.

Appendix B. Data Manipulation: Excerpts from R. Garza's EENG 699 Report

Data format deals with as many formats as there are sources. Different applications need a different setup for data, such as a single column of the data or two columns with spaces in between each row. The methods discussed in this section deal exclusively with ASCII to ASCII transformations.

There are three main methods available for this type of data manipulation. The first is to write a program in a computer language, such as C, that will do the necessary manipulations. The second method is to use several UNIX commands, AWK and SED, to format the data. The third method is to take the data into an application, such as MATLAB or MATHEMATICA, which will allow the necessary changes to be made. Due to the complexity of the first method, only the second and third methods will be discussed.

In order to give a real world example of the use of these methods, an example case will be set forth, with its solution shown in both of the two methods. Time series data, such as that for the stock market, often comes in a multicolumned, commented ASCII file. Many Algorithms, such as Casdagli's Deterministic Vs. Stochastic (DVS) or Sauer's Embedology, use a single column of data at a time, without spaces or comments. An example of a typical file is given below in the initdata file:

```
Rob-
    this is the data file you wanted
-Jim

881020    327.20
881021    332.55
881024    333.70
881025    332.05
881026    332.05
```

For use by the DVS algorithm, the first three lines and the first column of dates must be removed. This can be done in either of our two methods.

The AWK and SED commands are for use in the UNIX operating system only. These commands may be run in the command tool window or the terminal, dependent on the user's preference.

Examining the example file, there are two changes that have to be made. First, the first three lines must be removed. Second, the first columned needs to be removed. To remove the first three lines, use the SED command **sed '1,3d' filename**. Now remove the dates column. Choose to keep the second column by using the AWK command, **{print $2}**. The AWK command should be written in a short file, which I call **awkscr**. For the file **dataset**, set up a command tool with dataset's directory. The following code will be what the user needs to type in at the command line:

```
> sed '1,3d' dataset
> awk -f awkscr dataset>>dataset
```

Another method uses only the AWK command set. To impliment this, a short file needs to be written. The **changedata** file would contain the following:

```
#\!/bin/csh -f
awk -f awkscr initdata>>gooddata
```

The first line of the **changedata** code is set-up parameters. The second line contains the actual AWK command line routine.

The **awkscr** file would contain the following:

```
BEGIN{lines=0;}
{if(lines>2) print $2; lines=lines+1;}
END
```

With the initdata, changedata, and awkscr files all in the same directory, run the **changedata** file in the command tool or terminal by typing 'changedata' at the command line.

The file **gooddata** will be created, containing the following:

```
327.20
332.55
333.70
332.05
332.05
```

SED is often used to add or remove single characters, amongst other applications. For more on SED, see Lem Myers EENG 699 report.

This method employs a computer application to manipulate the data. The case presented here is done in Matlab. To run MATLAB on any of the sparc stations, type 'matlab' at the command line of a command tool.

First, the comment lines at the beginning of the file must be removed in a text editor. The file must then be read into Matlab. This is done by typing **load filename.xxx** at the Matlab prompt. This creates a variable 'filename' in the application. Type **diary on** at the prompt, and press return. This starts a screen dump of the command lines in Matlab. Type **x=sandp(:,2)** to keep only the second data column. Type **diary off** to end the screen dump, and exit matlab by typing 'quit' at the command line. Take the diary file into a text editor, using the 'load' command or double-clicking on the file in the file manager. Remove the extra lines at the beginning and end of the file using the 'cut' option in the editing menu. Highlight the words and symbols you want to remove, and select the 'cut' button. Rename the file, and the data is ready for the algorithm.

```
%detrend the data

load newsandp;
data=newsandp;

%Jim's Method (First Difference Method)
n=1100;

for i=2:n;
jim(i)=data(i)-data(i-1);
end;

jim =jim';
save newsandp1 jim -ascii;


%dbarr's method (First Ratio Method)

for j=2:n;
barr(j)=log(data(j)/data(j-1));
end;

barr =barr';
save newsandp2 barr -ascii;


%book method (First Image Method)

for k=2:n;
book(k)=(data(k)-data(k-1))/(data(k)+data(k-1));
end;

book =book';
save newsandp3 book -ascii;
```

```
%matlab m-file which creates training data for mlp nn

clear;

load closesh66v1.dat;
x=closesh66v1;

[a,n]=size(x);
m=n-3;
k=3;    %k is 2*f+1, where f is the fractal dimension

for i=1:m
for j=2:k+1;
train(i,j)=x(i+j-2);
end;
end;


for l=1:m-1

if train(l+1,4) > 0
train(l,1)=1;
else
train(l,1)=0;
end;

if train(l+1,4) == 0
train(l,1)=2;
else
k=k;
end;

end;

save closesh66v1.train  train -ascii
```

Appendix D. Curtis Martin's Non-Parametric Density Estimation Code

```
% CLASSIFY: Select thresholds and classify resubstitution and
%           leave-one-out discriminant values.
%
%     [R, L] = CLASSIFY(Lr1, Lr2, Ll1, Ll2, option)
%
% Inputs:  Lr1, Lr2: resubstitution discriminant values
%          Ll1, Ll2: leave-one-out discriminant values
%          option:   1 = threshold option 3
%                    2 = threshold option 4
%
% Outputs: R: Resubstitution error
%          L: Leave-one-out error

function [R, L] = classify(Lr1, Lr2, Ll1, Ll2, option)

% First get the minimum resubstitution error and threshold
fprintf(1,' Resubstitution...');
[R, t] = min_error(Lr1, Lr2);
fprintf(1,'done.\n');

% Now, depending on the option, get the minimum leave-one-out error
fprintf(1,' Leave one out...');
if option == 1
    L = sum(Ll1 > t) + sum(Ll2 < t);
else
    n1 = size(Ll1, 2);
    n2 = size(Ll2, 2);
    L = 0;
fprintf(1,'Class 1...');
    for i = 1:n1,
        [err, t] = min_error(Ll1([1:i-1 i+1:n1]), Ll2);
        if Ll1(i) > t
            L = L + 1;
        end  % if Ll1(i) > t
    end  % for i
fprintf(1,'Class 2...');
    for i = 1:n2,
        [err, t] = min_error(Ll1, Ll2([1:i-1 i+1:n1]));
```

```
        if Ll2(i) < t
            L = L + 1;
        end  % if Ll2(i) < t
    end  % for i
end  % if option == 1
fprintf(1,'done.\n');
```

```
% COMPUTE_DISTANCES:  Compute K distances between samples
%                     in X1 and X2.
%
%     [D11, D12, D21, D22] = COMPUTE_DISTANCES(X1, X2, invS1, invS2)
%
%   D11 = class 1 distances for samples of class 1
%   D12 = class 2 distances for samples of class 1
%   D21 = class 1 distances for samples of class 2
%   D22 = class 2 distances for samples of class 2
%
% invS1 and invS2 are the covariance matrix inverses
% (maybe estimated).
% X1 and X2 have one observation per column.

function [D11, D12, D21, D22] = compute_distances(X1, X2,
 invS1, invS2)

% Number of columns is the number of samples
N1 = size(X1,2);
N2 = size(X2,2);

% Allocate space
D11 = zeros(N1,N1);
D12 = zeros(N1,N2);
D21 = zeros(N2,N1);
D22 = zeros(N2,N2);

% Calculate intra-class distances:
  % Class 1:
for i = 1:N1-1
    for j = i+1:N1
D11(i,j) = (X1(:,j) - X1(:,i))' * invS1 * (X1(:,j) - X1(:,i));
D11(j,i) = D11(i,j);
    end
end

  % Class 2:
for i = 1:N2-1
    for j = i+1:N2
D22(i,j) = (X2(:,j) - X2(:,i))' * invS2 * (X2(:,j) - X2(:,i));
D22(j,i) = D22(i,j);
    end
```

```
end

% Calculate inter-class distances:
for i = 1:N1              % Rows for D12, Columns for D21
    for j = 1:N2          % Columns for D12, Rows for D21
% Pull samples out of matrices only once
v = X2(:,j) - X1(:,i);
D12(i,j) = v' * invS1 * v;

% These could be (-v), but there's no reason for it. (note (j,i))
D21(j,i) = v' * invS2 * v;
    end
end
```

```
% MIN_ERROR: Select threshold which gives the minimum error
%            for classifying two sets of discriminants.
%            This is designed to be used for any number of
%            elements in each set.  In the case of several
%            minimum errors, the threshold that yields the
%            same number of misclassifications from set 1 and
%            set 2 is selected.
%
%     [ERROR, THRESHOLD] = MIN_ERROR(SET1, SET2)
%
% Inputs:  SET1, SET2: sets of discriminants from classes 1
%    & 2
%
% Outputs: ERROR: Minimum error obtainable
%          THRESHOLD: "Best?" threshold yielding the minimum error

function [error, threshold] = min_error(set1, set2)

big = realmax - realmin;
search = 0;

% Eliminate infinities from the search.
infs = find(set1==inf);
set1(infs) = big * ones(size(infs));
infs = find(set1==(-inf));
set1(infs) = -big * ones(size(infs));
infs = find(set2==inf);
set2(infs) = big * ones(size(infs));
infs = find(set1==(-inf));
set2(infs) = -big * ones(size(infs));

a = min(set1);
b = max(set1);
c = min(set2);
d = max(set2);

if b < c
    threshold = 0.5 * b + 0.5 * c;
    error = 0;
    return;
end  % if
```

```
if (min([a b c d]) == c) & (max([a b c d]) == b)
    %pick c or b at random.
    choice = round(rand);
    if choice == 0
        threshold = b + realmin;
    else
        threshold = c - realmin;
    end  % if choice
    error = sum(set1 > threshold) + sum(set2 < threshold);
    fprintf(1,'Bad decision rule: a=%d, b=%d, c=%d, d=%d\n',a,b,c,d);
    return;
end  % if

if (a < c & c < d & d < b) | ( c < a & a < b & b < d )
    lo = a;
    hi = d;
elseif (a < c & c < b & b < d)
    lo = c;
    hi = b;
else
    % What's left??? Write a message.write out a, b, c, and d
    fprintf(1,'Equal condition: a=%d, b=%d, c=%d, d=%d\n',a,b,c,d);
    threshold = .25 * a + .25 * b + .25 * c + .25 * d;
    error = -1;
    return;
end  % if

% go fish (in a limited pool)
pool = [set1(set1>=lo & set1<=hi) set2(set2>=lo & set2<=hi)];

% Get one copy of each distinct value in temp
temp(1) = lo;
n = size(pool,2);
for i = 2:n,
    temp(i) = min(pool(pool>temp(i-1)));
    if temp(i) == hi
        break
    end % if
end  % for
n = size(temp,2);

err1 = zeros(1, n+1);
```

```
err2 = zeros(1, n+1);

% Count the errors for each guess
for i=1:n,
    err1(i) = sum(set1 >= temp(i));
    err2(i) = sum(set2 < temp(i));
end  % for
err1(n+1) = sum(set1 > temp(n));
err2(n+1) = sum(set2 <= temp(n));

% Find the minimum total error
total = err1 + err2;
error = min(total);
index = find(total == error);
if size(index,2) == 1
    tidx = index;
else
    diff = abs(err1(index) - err2(index));
    mindif = min(diff);
    didx = find(diff == mindif);
    nminds = size(didx, 2);
    if nminds == 1
        tidx = index(didx);
    else
        choice = round((nminds-1) * rand) + 1;
        tidx = index(didx(choice));
    end  % if nminds
end  % if size(index,2)

if tidx > 1 & tidx <= n
    threshold = 0.5 * temp(tidx) + 0.5 * temp(tidx-1);
elseif tidx == 1
    lower = max([max(set1(set1<temp(1))) max(set2(set2<temp(1)))]);
    if size(lower,2) == 0
        threshold = temp(1) - realmin;
    else
        threshold = 0.5 * temp(1) + 0.5 * lower;
    end  % if size(lower, 2)
else
    higher = min([min(set1(set1>temp(n))) min(set2(set2>temp(n)))]);
    if size(higher,2) == 0
        threshold = temp(n) + realmin;
```

```
    else
        threshold = 0.5 * temp(n) + 0.5 * higher;
    end  % if size(higher, 2)
end  % if tidx...
```

```
% PKNN:  Run Parzen and kNN procedure for X1, X2 10 times.
%
%     [Rp, Lp, Rk, Lk] = PKNN(X1, X2, h, k, opt)
%
% Inputs:  X1, X2:  data sets (n x N1 and n x N2)
%          h, k:    values to use for h and k
%          opt:     1 = threshold option 3
%                   2 = threshold option 4
%
%
% All h's must be greater than zero, and k must be between 2
% and min(N1, N2)-1
%
% Outputs: Rp, Lp: Parzen R and L errors for each test
%          (one test per row)
%          Rk, Lk: k-NN R and L errors for each test

function [Rp, Lp, Rk, Lk] = pknn(X1, X2, h, k, opt)

[n1, N1] = size(X1);
[n2, N2] = size(X2);
if n1 ~= n2
    fprintf(2, 'Data sets X1 and X2 must have same number
of rows (features)\n');
    return;
end  % if

% Keep this value on hand
dim = n1;
ntests = 5;

Rp = zeros(ntests, size(h,2));
Lp = zeros(ntests, size(h,2));
Rk = zeros(ntests, size(k,2));
Lk = zeros(ntests, size(k,2));

fprintf(1, 'Threshold Option = %d \n', opt);
fprintf(1, 'Performing %d independent tests: \n', ntests);

for test = 1:ntests,

    fprintf(1,' Test #%d:\n', test);
```

```
testsamps = test:ntests:N1;
t1 = [1, testsamps+1];
t2 = [testsamps-1, N1];
others = [];
for i = 1:size(t1,2),
    others = [others, t1(i):t2(i)];
end  % for i

x1 = X1(:, testsamps);
iS1 = cov(X1(:,others)');

testsamps = test:ntests:N2;
t1 = [1, testsamps+1];
t2 = [testsamps-1, N2];
others = [];
for i = 1:size(t1,2),
    others = [others, t1(i):t2(i)];
end  % for i

x2 = X2(:, testsamps);
iS2 = cov(X2(:,others)');

clear testsamps others t1 t2

n1 = size(x1, 2);
n2 = size(x2, 2);
fprintf(1, '  %d Class 1 samples, %d Class 2 samples\n', n1, n2);

fprintf(1, '  Estimating and inverting covariance matrices
     ...\n');
detratio = -0.5 * log(det(iS2)/det(iS1));
iS1 = inv(iS1);
iS2 = inv(iS2);

fprintf(1, '  Computing distances ...\n');

[d11, d12, d21, d22] = compute_distances(x1, x2, iS1, iS2);

fprintf(1, '  Classifying (Parzen) ...\n');
Rerr = [];
Lerr = [];
```

```
for r = h,

    % Compute sums:
    temp = -0.5 / r^2;
    s11 = sum(exp(temp * d11));
    s12 = sum(exp(temp * d12));
    s21 = sum(exp(temp * d21));
    s22 = sum(exp(temp * d22));

    lr1 = detratio - log((n2 * s11) ./ (n1 * s21));
    lr2 = detratio - log((n2 * s12) ./ (n1 * s22));

    ll1 = detratio - log((n2 * (s11 - 1)) ./ ((n1 - 1) * s21));
    ll2 = detratio - log(((n2 - 1) * s12) ./ (n1 * (s22 - 1)));

    [rerr, lerr] = classify(lr1, lr2, ll1, ll2, opt);

    Rerr = [Rerr, rerr];
    Lerr = [Lerr, lerr];

end  % for r

Rp(test,:) = 100 * Rerr / (n1+n2);
Lp(test,:) = 100 * Lerr / (n1+n2);


fprintf(1,'  Classifying (k-NN) ...\n');

% sort distances
d11 = sort(d11);
d12 = sort(d12);
d21 = sort(d21);
d22 = sort(d22);

tr = detratio + log(n1/n2);
tl1 = detratio + log((n1-1)/n2);
tl2 = detratio + log(n1/(n2-1));
Rerr = [];
Lerr = [];

for i = k,
```

```
        lr1 = tr + 0.5 * dim * log(d11(i,:) ./ d21(i,:));
        lr2 = tr + 0.5 * dim * log(d12(i,:) ./ d22(i,:));

        ll1 = tl1 + 0.5 * dim * log(d11(i+1,:) ./ d21(i,:));
        ll2 = tl2 + 0.5 * dim * log(d12(i,:) ./ d22(i+1,:));

        [rerr, lerr] = classify(lr1, lr2, ll1, ll2, opt);

        Rerr = [Rerr, rerr];
        Lerr = [Lerr, lerr];

    end  % for i

    Rk(test,:) = 100 * Rerr / (n1+n2);
    Lk(test,:) = 100 * Lerr / (n1+n2);

end  % for test
```

Appendix E. Jim Stright's DVS Algorithm C Code

The Casdagli Algorithm is designed to do time series prediction by determining the underlying principles of the time series. The algorithm will determine whether the time series is low dimensional deterministic or high dimensional stochastic, to better estimate the predicted value.

The x vector is an observed scalar time series generated from a D dimensional attractor of a deterministic dynamic system with d degrees of freedom.

Choose an embedding dimension m, a delay time tau, and a forecasting time T. The dimension, m, should be the last dimension which significantly minimizes the root-mean-square error. Tau depends on the length of the delay between points that you want. Finally, T depends on the length of prediction that you want.

Calculate all of the nearest neighbors for each time series value of newsandp.

Find the standard deviation of the time series, future use in error calculations. Then find the maximum number of vectors which will be compared for nearness, and calculate the error matrix e(k)(i).

After finding the error matrix for the particular i, repeat the previous steps for all i. This will allow for the computation of the normalized root-mean square (RMS) forecasting error Em(k). The program then outputs the number of nearest neighbors and its forecasting error for user analysis. The predicted value is also output at this time. This output is written to the file casdata.

The remainder of the code defines vectors, and defines error sequences for the program.

The following code calculates the relative mean-square error for various numbers of nearest neighbors when fitting a time series to a certain embedology dimension. This code was written by Capt. Jim Stright in the course of his pHD work here at AFIT.

```
/* This program, casdagli7.c, implements the forecasting
   algorithm described on page 307 of Casdagli's article
   "Chaos and Deterministic versus Stochastic Non-linear
   Modelling."  It was written in May 1993 by Jim Stright.
   Casdagli7.c also provides a prediction of a single value
   beyond the end of the data used for testing (an unknown).
   It does so using the best m & k as found from previous
   runs of this program. Usually Nt=0 is used in the
   prediction mode, with Nf the number of time series values
   assumed known.   Many of the subroutines are taken from
   Press et al, "Numerical Recipes in C."  As a predictor,
   casdagli7.c implements "DVS prediction." */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define TINY 1.0e-20
double *dvector();
int *ivector();
double **dmatrix();
double moment();
double ludcmp();
double lubksb();
double sort2();
void free_dvector(),free_dmatrix(),free_ivector();

void main(void)
{
FILE *fp1, *fp2;          /* fp1 is newsandp (input);
    fp2 is casdata */
int m=6;                  /* embedding dim */
int tau = 1;              /* delay time */
int T = tau;              /* forecasting time;
           not necessarily tau!! */
int kbest = 2*(m+1);      /* replace with best k,
    else use 2*(m+1) */
int i,j,ctr1,ctr2,ctr3;   /* counters */
int row,col,l;            /* more counters */
int k;                    /* nbr nearest neighbors */
int klast = 0;            /* This counter is at the nbr (+2*(m+1))
    of the last nearest neighbor incorporated
                                  in the A matrix */
```

```
int Nf = 2500;          /* nbr of time series values
                           in fitting set */
int Nt = 150;           /* nbr of time series values
                           in testing set */
int Ns = 1;             /* spacing of the sampled
                           delay vectors */
int n;                  /* required for call to
                           "moment" */
int FLAG = 0;           /* used in ludcmp for "too
                           large" check */
int kexp = 0;           /* counter for exponential
                           spacing of k's */
double kbase = 2.0;     /* base for exponential
                           spacing of k's  */

double ave,adev,sigma,svar,skew,curt;
        /* all of these required for call "moment" although
           only sigma is used in casdagli7.c; see Press Ed 2,
           p.613 */
double *ave_ptr=&ave,*adev_ptr=&adev,*sigma_ptr=&sigma;
double *svar_ptr=&svar,*skew_ptr=&skew,*curt_ptr=&curt;
double *x;

double **A,**Alud,**d,*dhold,*alpha,*b,dnr;
       /* Alud is repeatedly destroyed by ludcmp, dnr is
          Press's d, p.46  */
int *indx;
int *nbrtested;
double **xhat,**e,*Em,errsum;
x = dvector(1,Nf+Nt);
indx = ivector(1,m+1);
nbrtested = ivector(0,Nf-T-(m-1)*tau-2*(m+1));
A = dmatrix(1,m+1,1,m+1);
Alud = dmatrix(1,m+1,1,m+1);
d = dmatrix(Nf,Nf+Nt,1,Nf-T-(m-1)*tau);
/* d[i][1] is the distance from vector x[i] to vector
           x[1+(m-1)*tau];
   d[i][2] is the distance from vector x[i] to vector
           x[1+(m-1)*tau+1];
 ...
   d[i][Nf-T-(m-1)*tau] is distance from vector x[i]
           to vector x[Nf-T], before a swap for nearness is
```

```
   performed. */

dhold = dvector(1,Nf-T-(m-1)*tau);
alpha = dvector(1,m+1);
b = dvector(1,m+1);

xhat = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt+T);
e = dmatrix(0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt);
Em = dvector(0,Nf-T-(m-1)*tau-2*(m+1));

/* open newsandp for input */
if ((fp1 = fopen("newsandp","r")) == NULL) {
printf("Cannot open file newsandp\n");
exit(1);
}

/* open casdata for output */
if ((fp2 = fopen("casdata", "w")) == NULL) {
printf("Cannot open file casdata\n");
exit(1);
}

/* read in the time series data */
for (ctr1=1; ctr1<=Nf+Nt; ctr1++) {
fscanf(fp1, "%lf", &x[ctr1]);
}

/* compute distances d[i][j] and load d matrix with
           nearness indices */
for (i=Nf; i<=Nf+Nt; i++) {
for (j=1; j<=Nf-T-(m-1)*tau; j++) { /* see
            indexing note below */
d[i][j] = fabs(x[i] - x[j+(m-1)*tau]);
for (ctr1=tau; ctr1<=(m-1)*tau; ctr1=ctr1+tau) {
if (fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1])
                                    > d[i][j]) {
  d[i][j] = fabs(x[i-ctr1]-x[j+(m-1)*tau-ctr1]);
}
}
/* dist d[i][j] between vctrs x[i] &
                           x[j+(m-1)*tau] is fixed */
}
```

```
       /* the distances d[i][j] are now established for all j */

       /* initialize the index-swap vector dhold */
for (ctr2=1; ctr2<=Nf-T-(m-1)*tau; ctr2++) {
dhold[ctr2] = ctr2 + (m-1)*tau;
            /* now the contents of dhold[1], eg, is 1+(m-1)*tau   */
}

/* Sort the contents of the vector d[i] and
      simultaneously sort the vector dhold into ascending
                  order of nearness of vectors to x[i];
   see Press Ed 2, page 334. */

sort2(Nf-T-(m-1)*tau, d[i], dhold);

/* replace contents of vector d[i] with indices of
           vectors arranged in ascending order of nearness to
                  x[i] */

for (ctr3=1; ctr3<=Nf-T-(m-1)*tau; ctr3++) {
d[i][ctr3] = dhold[ctr3];
}
        /* Now the contents of vector d[i] is the set of indices of
           vectors compared for nearness to vector x[i], arranged in
           ascending order of nearness to x[i].  eg, d[i][1] is the
           index of the vector nearest x[i]. */
}

/* Find standard dev sigma for the time series; see Press 2,
           page 613.  */
n = Nf+Nt;
moment(x,n,ave_ptr,adev_ptr,sigma_ptr,svar_ptr,skew_ptr,curt_ptr);

fprintf(fp2,"Data output from program casdagli7.c\n");
fprintf(fp2,"m=%2d\n",m);
fprintf(fp2,"Nf = %d\n",Nf);
fprintf(fp2,"Nt = %d\n",Nt);
fprintf(fp2,"T = tau = %d\n",T);
fprintf(fp2,"average data value ave = %2.5f\n",ave);
fprintf(fp2,"data standard deviation sigma = %2.5f\n",sigma);

/* Initialize the max nbr of vectors to be compared for nearness  */
```

```
k = 0;
kexp = 0;
while (k <= Nf-T-(m-1)*tau-2*(m+1)) {
nbrtested[k] = (Nt-T+1)/Ns;  /* recall int division trucates  */
k = (int) pow(kbase,kexp);
kexp = kexp + 1;
}

/* Establish the error matrix e[k][i]                                  */
for (i=Nf; i<=Nf+Nt; i++) {
/* Initialize the A matrix at k=2*(m+1) */
/* First the diagonal entries: */
A[1][1] = 2*(m+1);
Alud[1][1] = A[1][1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
A[ctr1][ctr1] = 0.0;
for (ctr2=1; ctr2<=2*(m+1); ctr2++) {
A[ctr1][ctr1] = A[ctr1][ctr1]
+ x[(int)(d[i][ctr2])-(ctr1-2)*tau]
                * x[(int)(d[i][ctr2])-(ctr1-2)*tau];
}
Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
A[1][col] = 0.0;
for (l=1; l<=2*(m+1); l++) {
A[1][col] = A[1][col] + x[(int)(d[i][l])-(col-2)*tau];
}
Alud[1][col] = A[1][col];
A[col][1] = A[1][col];
            Alud[col][1] = A[col][1];
}

/* Now initialize the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
for (col=row+1; col<=(m+1); col++) {
A[row][col] = 0.0;
for (l=1; l<=2*(m+1); l++) {
A[row][col] = A[row][col]
+ x[(int)(d[i][l])-(row-2)*tau]
```

```
                    * x[(int)(d[i][l])-(col-2)*tau];
}
Alud[row][col] = A[row][col];
A[col][row] = A[row][col];
                Alud[col][row] = A[col][row];
}
}

/* And last, initialize the b vector, and its equal alpha vector;
   alpha gets replaced with the proper solution when one solves
   A*alpha = b as A*x=b=alpha; see Press, page 44.                */
b[1] = 0.0;
for (l=1; l<=2*(m+1); l=l+1) {
b[1] = b[1] + x[(int)(d[i][l])+T];
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
b[row] = 0.0;
for (l=1; l<=2*(m+1); l++) {
b[row] = b[row]
+ x[(int)(d[i][l])-(row-2)*tau]
* x[(int)(d[i][l])+T];
}
alpha[row] = b[row];
}

/* Go after the e[k][i]; may easily change the k indexing to sample
   k's at exponentially spaced intervals.  In what follows, k
   equals the nbr of neighbors nearest x[i] minus 2*(m+1).       */

klast = 0;
k=0;
kexp = 0;
while (k <= Nf-T-(m-1)*tau-2*(m+1)) {

/* Nf-T-(m-1)*tau is the nbr of nghbrs in fitting set whose
   "predicted" value is <= Nf                                    */

/* Update the A matrix */

/* First update the diagonal entries */
A[1][1] = 2*(m+1) + k;
```

```
Alud[1][1] = A[1][1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
for (ctr2=1; ctr2<=(k-klast); ctr2++) {
A[ctr1][ctr1] = A[ctr1][ctr1]
+ x[(int)(d[i][2*(m+1)+klast+ctr2])-(ctr1-2)*tau]
* x[(int)(d[i][2*(m+1)+klast+ctr2])-(ctr1-2)*tau];
}
Alud[ctr1][ctr1] = A[ctr1][ctr1];
}

/* Now update the first row (and first column) entries: */
for (col=2; col<=(m+1); col++) {
for (l=1; l<=(k-klast); l++) {
A[1][col] = A[1][col]
    + x[(int)(d[i][2*(m+1)+klast+l])-(col-2)*tau];
}
Alud[1][col] = A[1][col];
A[col][1] = A[1][col];
Alud[col][1] = A[col][1];
}

/* Now update the off-diag, off-first-row-or-col entries: */
for (row=2; row<=m; row++) {
for (col=row+1; col<=(m+1); col++) {
for (l=1; l<=(k-klast); l++) {
A[row][col] = A[row][col]
+ x[(int)(d[i][2*(m+1)+klast+l])-(row-2)*tau]
* x[(int)(d[i][2*(m+1)+klast+l])-(col-2)*tau];
}
Alud[row][col] = A[row][col];
A[col][row] = A[row][col];
Alud[col][row] = A[col][row];
}
}

/* Finally, update the b vector: */
for (l=1; l<=(k-klast); l++) {
b[1] = b[1] + x[(int)(d[i][2*(m+1)+klast+l])+T];
}
alpha[1] = b[1];
for (row=2; row<=(m+1); row++) {
for (l=1; l<=(k-klast); l++) {
```

```
b[row] = b[row]
+ x[(int)(d[i][2*(m+1)+klast+l])-(row-2)*tau]
* x[(int)(d[i][2*(m+1)+klast+l])+T];
}
alpha[row] = b[row];
}

klast = k;

/* Solve the normal eqtns for alpha[1] thru alpha[m+1] */
ludcmp(Alud,FLAG,m+1,indx,&dnr);

if (FLAG==1) {
alpha[1] = 1000001;
FLAG=0;}
else {
lubksb(Alud,m+1,indx,alpha);
}

/* alpha[1],alpha[2],... are now optimum in Casdagli's eqtn 5, if
   the normal equations admit a solution.  Otherwise set alpha[1]
   = x[i+T], alpha[2]=alpha[3]=...=alpha[m+1]=0, so that
   xhat[k][i+T] = x[i+T], the exact data value.  Also, decrement
   nbrtested so this unusual event isn't included in Em[k].     */

for (ctr1=1; ctr1<=(m+1); ctr1++) {
if ((fabs(alpha[ctr1]) >  1000000)||
(alpha[ctr1] == HUGE_VAL)) {
nbrtested[k] = nbrtested[k]-1;
fprintf(fp2,"Error at 1\n");
alpha[1] = x[i+T];
for (ctr2=2; ctr2<=(m+1);ctr2++) {
alpha[ctr2] = 0.0;
}
break;
}
}

xhat[k][i+T] = alpha[1];
for (ctr1=2; ctr1<=(m+1); ctr1++) {
xhat[k][i+T] = xhat[k][i+T] + alpha[ctr1]*x[i-(ctr1-2)*tau];
}
```

```
/* xhat[k][i+T] has now been established */


if (i <= Nf+Nt-T)
e[k][i] = fabs(xhat[k][i+T] - x[i+T]);
else
e[k][i] = 0.0;

k = (int) pow(kbase,kexp);
kexp = kexp + 1;
}   /* closes the loop over k's */
}  /* closes the loop over i's */

k = 0;
kexp = 0;
while (k <= Nf-T-(m-1)*tau-2*(m+1)) {
errsum = 0.0;
for (i=Nf; i<=Nf+Nt-T; i++) {
errsum = errsum + e[k][i]*e[k][i];
}
Em[k] = (sqrt(errsum/nbrtested[k]))/sigma;

fprintf(fp2, "%d", 2*(m+1)+k);  /* Output nbr of nearest nghbrs  */
fprintf(fp2,"     ");
fprintf(fp2, "%1.6f\n", Em[k]);  /* Output forecasting error  */

k = (int) pow(kbase,kexp);
kexp = kexp + 1;
}

fprintf(fp2, "Predicted value at time %d is %f\n",
Nf+Nt+T,xhat[kbest-2*(m+1)][Nf+Nt+T]);

free_dvector(x,1,Nf+Nt);
free_dvector(dhold,1,Nf-T-(m-1)*tau);
free_dvector(alpha,1,m+1);
free_dvector(b,1,m+1);
free_dvector(Em,0,Nf-T-(m-1)*tau-2*(m+1));
free_ivector(indx,1,m+1);
free_ivector(nbrtested,0,Nf-T-(m-1)*tau-2*(m+1));
free_dmatrix(A,1,m+1,1,m+1);
free_dmatrix(Alud,1,m+1,1,m+1);
```

```
free_dmatrix(d,Nf,Nf+Nt-T,1,Nf-T-(m-1)*tau);
free_dmatrix(xhat,0,Nf-T-(m-1)*tau-2*(m+1),Nf+T,Nf+Nt);
free_dmatrix(e,0,Nf-T-(m-1)*tau-2*(m+1),Nf,Nf+Nt-T);
fclose(fp1);
fclose(fp2);
}

double moment(data,n,ave,adev,sdev,svar,skew,curt)
int n;
double *data,*ave,*adev,*sdev,*svar,*skew,*curt;
{
int i,j;
double s,p;
void nrerror();

if (n <= 1) nrerror("n must be at least 2 in MOMENT");

s=0.0;
for (j=1;j<=n;j++) s += data[j];
*ave=s/n;
*adev=(*svar)=(*skew)=(*curt)=0.0;
for (j=1;j<=n;j++) {
*adev += fabs(s=data[j]-(*ave));
*svar += (p=s*s);
*skew += (p *= s);
*curt += (p *= s);
}
*adev /= n;
*svar /= (n-1);
*sdev=sqrt(*svar);
if (*svar) {
*skew /= (n*(*svar)*(*sdev));
*curt=(*curt)/(n*(*svar)*(*svar))-3.0;
} else nrerror("No skew/kurtosis when variance = 0 (in MOMENT)");
}

double ludcmp(a,FLAG,n,indx,d)
int FLAG,n,*indx;
double **a,*d;
{
int i,imax,j,k;
double big,dum,sum,temp;
```

```
double *vv,*dvector();
void nrerror(),free_dvector();

vv=dvector(1,n);
*d=1.0;
for (i=1;i<=n;i++) {
big=0.0;
for (j=1;j<=n;j++)
if ((temp=fabs(a[i][j])) > big) big=temp;
if (big == 0.0){
FLAG = 1;
return;
}
vv[i]=1.0/big;
}
for (j=1;j<=n;j++) {
for (i=1;i<j;i++) {
sum=a[i][j];
for (k=1;k<i;k++) sum -= a[i][k]*a[k][j];
a[i][j]=sum;
}
big=0.0;
for (i=j;i<=n;i++) {
sum=a[i][j];
for (k=1;k<j;k++)
sum -= a[i][k]*a[k][j];
a[i][j]=sum;
if ( (dum=vv[i]*fabs(sum)) >= big) {
big=dum;
imax=i;
}
}
if (j != imax) {
for (k=1;k<=n;k++) {
dum=a[imax][k];
a[imax][k]=a[j][k];
a[j][k]=dum;
}
*d = -(*d);
vv[imax]=vv[j];
}
indx[j]=imax;
```

```
if (a[j][j] == 0.0) a[j][j]=TINY;
if (j != n) {
dum=1.0/(a[j][j]);
for (i=j+1;i<=n;i++) a[i][j] *= dum;
}
}
free_dvector(vv,1,n);
}

double lubksb(a,n,indx,b)
double **a,b[];
int n,*indx;
{
int i,ii=0,ip,j;
double sum;

for (i=1;i<=n;i++) {
ip=indx[i];
sum=b[ip];
b[ip]=b[i];
if (ii)
for (j=ii;j<=i-1;j++) sum -= a[i][j]*b[j];
else if (sum) ii=i;
b[i]=sum;
}
for (i=n;i>=1;i--) {
sum=b[i];
for (j=i+1;j<=n;j++) sum -= a[i][j]*b[j];
b[i]=sum/a[i][i];
}
}

double sort2(n,ra,rb)
int n;
double ra[],rb[];
{
int l,j,ir,i;
double rrb,rra;

l=(n >> 1)+1;
ir=n;
for (;;) {
```

```
if (l > 1) {
rra=ra[--l];
rrb=rb[l];
} else {
rra=ra[ir];
rrb=rb[ir];
ra[ir]=ra[1];
rb[ir]=rb[1];
if (--ir == 1) {
ra[1]=rra;
rb[1]=rrb;
return;
}
}
i=l;
j=l << 1;
while (j <= ir) {
if (j < ir && ra[j] < ra[j+1]) ++j;
if (rra < ra[j]) {
ra[i]=ra[j];
rb[i]=rb[j];
j += (i=j);
}
else j=ir+1;
}
ra[i]=rra;
rb[i]=rrb;
}
}

void nrerror(error_text)
char error_text[];
{
void exit();

fprintf(stderr,"Numerical Recipes run-time error...\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...now exiting to system...\n");
exit(1);
}
```

```
int *ivector(nl,nh)
int nl,nh;
{
int *v;

v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
if (!v) nrerror("allocation failure in ivector()");
return v-nl;
}

double *dvector(nl,nh)
int nl,nh;
{
double *v;

v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
if (!v) nrerror("allocation failure in dvector()");
return v-nl;
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
int i;
double **m;

m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
if (!m) nrerror("allocation failure 1 in dmatrix()");
m -= nrl;

for(i=nrl;i<=nrh;i++) {
m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
m[i] -= ncl;
}
return m;
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
```

```
free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
free((char*) (v+nl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
int i;

for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
free((char*) (m+nrl));
}
```

```
%%%%%%%% tlsprony.m - Run this code to run the algorithm
%%%%%%%% Input at command line : [phase,magnitude,amplitude]=
%%%%%%%% tlsprony(\# of points, model \#, 1/3 \# of points, 0);

function [p,A,sh]=tlsprony(s,L,ns,constraint)

p=backsolv(s,L,ns,constraint);

[p,A,sh]=ampsolv(s,p,ns);


%%%%%%%% backsolv.m - A Component File for Use with the tlsprony.m

function p=backsolv(s,L,ns,constraint)

%
%  data should be in column form, each new look in a new column.
%

[N,looks]=size(s);
nrows=N-L;

Q=zeros(looks*nrows,L+1);
for i=1:looks,
  Q((i-1)*nrows+1:i*nrows,:)=hankel(s(1:nrows,i),s(nrows:N,i));
end

b=svdtls(Q,ns);
p=roots(flipud([1;b]));

if constraint==1,
   p=p./abs(p);
end;

%%%%%%%% svdlts.m - A Component File for Use with the backsolv.m
function b=svdtls(Q,ns)

[dummy,n]=size(Q);
[U,E,V]=svd(Q,0);
```

```
V2p=V(2:n,ns+1:n);
c=V(1,ns+1:n).';
b=V2p*conj(c)/(c'*c);
```

```
%%%%%%%% ampsolv.m - A Component File for Use with the tlsprony.m

function [p,A,sh]=ampsolv(s,p,ns)

[N,dummy]=size(s);

p(find(abs(p)>(1.2)|abs(p)<1/1.2))=[];

if length(p)==0,
  A=[];
  sh=zeros(size(s));
else
 [A,dummy]=lsamp(s,p);

 [p,A]=pkeep(p,A,ns,N);
 [A,sh]=lsamp(s,p);

end

%%%%%%%% lsamp.m - A Component File for Use with the ampsolve.m

function [A,sh]=lsamp(s,p)

[N,dummy]=size(s);

P=ones(N,length(p));
for n=1:N-1,
  P(n+1,:)=P(n,:).*p.';
end

A=P\s;

sh=P*A;

%%%%%%%% pkeep.m - A Component File for Use with the ampsolve.m

function [r,a]=pkeep(p,A,ns,N)

e=energy(p,A,N);

[e,i]=sort(e);
```

```
i=flipud(i);
if ns <= length(i),
 i=i(1:ns);
else
 i=i(1:length(i));
end
r=p(i,:);
a=A(i,:);
```

```
%%%%%%%% energy.m - An Additional File for Use with the tlsprony.m
%%%%%%%% if the energy is to be included in the determination,
%%%%%%%% which is the suggested configuration.
%%%%%%%% Input at command line :
%%%%%%%% [phase,magnitude,amplitude,energy]=
%%%%%%%% tlsprony(\# of points, model \#, 1/3 \# of points, 0);

function e=energytest(p,A,N);

[dummy,looks]=size(A);
[np,psets]=size(p);
absp=abs(p);

if looks==1,
  ea=abs(A).^2;
else
  ea=sum(abs(A).^2')';
end

if psets==1,
  ep=(1-absp.^(2*N)+N*(absp==1))./(1-absp.^2+(absp==1));
else
  ep=prod(((1-absp.^(2*ones(np,1)*N)+ones(np,1)*N.*(absp==1))./ ...
          (1-absp.^2+(absp==1)))')';
end

e=ea.*ep;
```

Appendix G. Bayes Bounding Curves

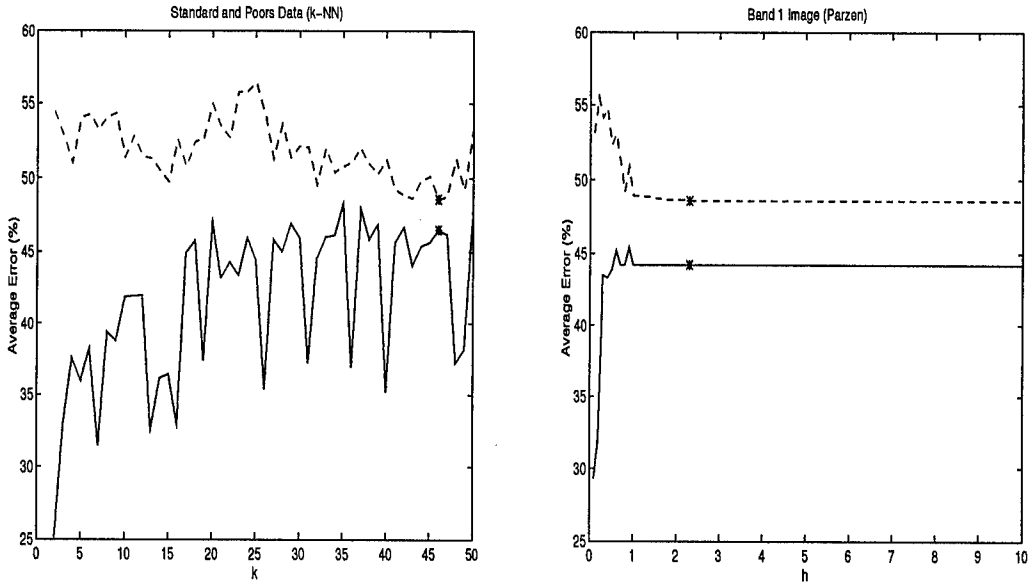


Figure 40. Bayes Bounding of Raw SandP data, m=1

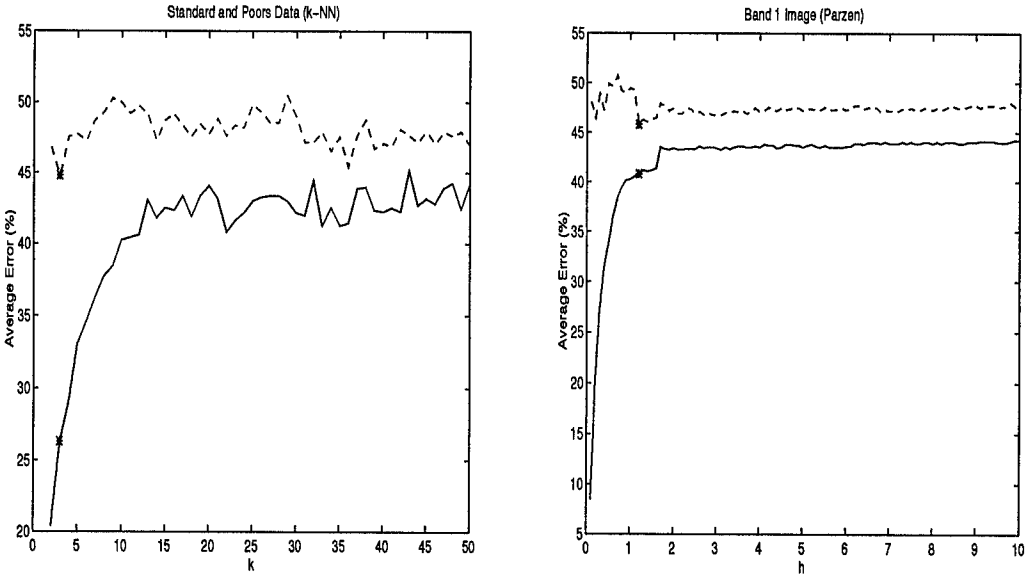


Figure 41. Bayes Bounding of Raw SandP data, m=2

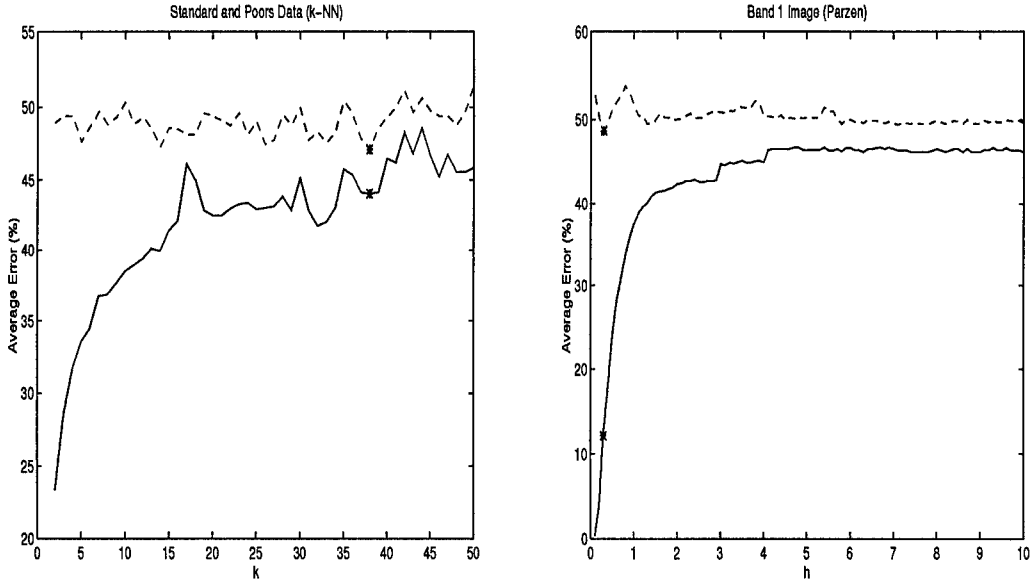


Figure 42. Bayes Bounding of Raw SandP data, m=3

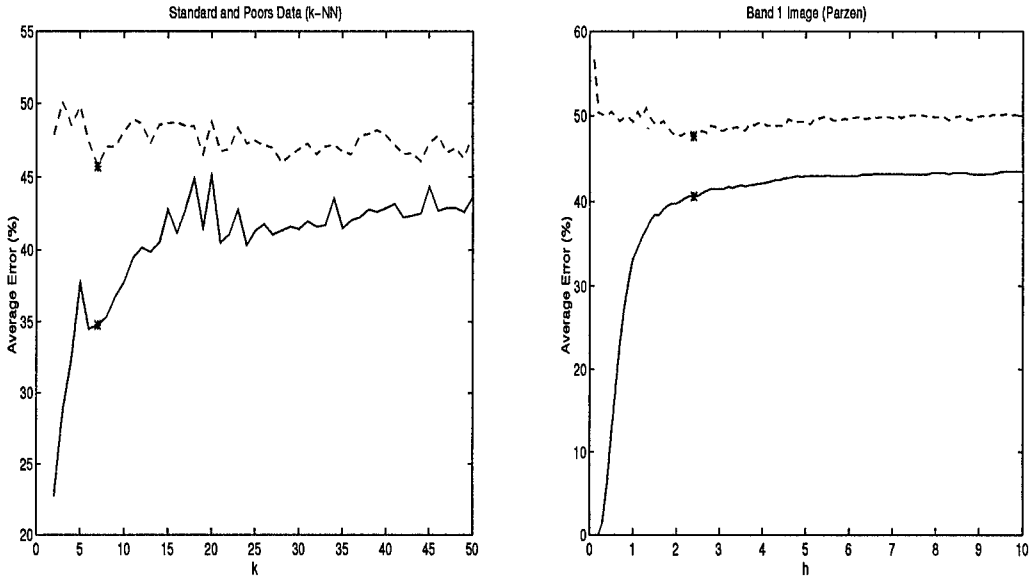


Figure 43. Bayes Bounding of Raw SandP data, m=4

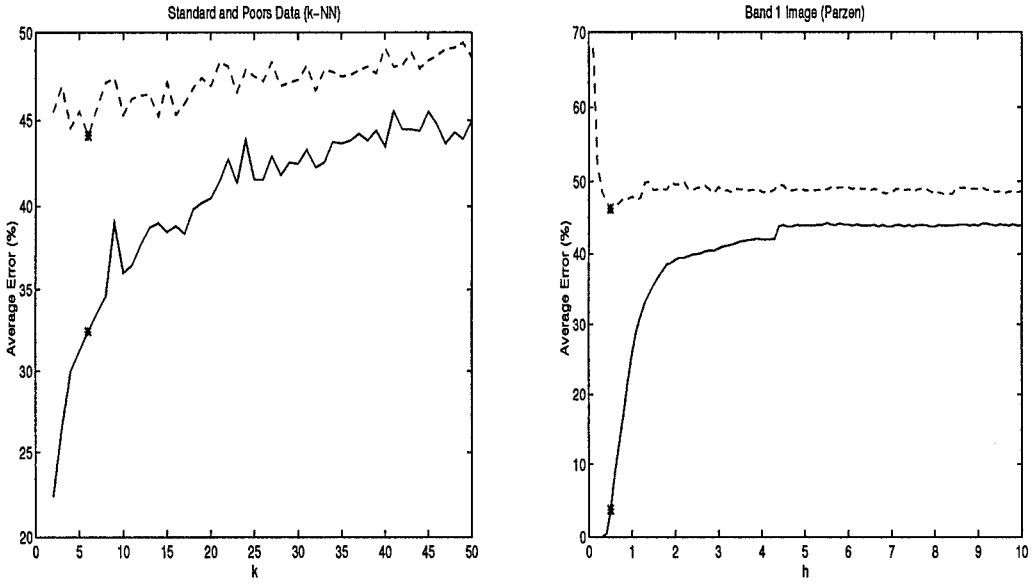


Figure 44. Bayes Bounding of Raw SandP data, m=5

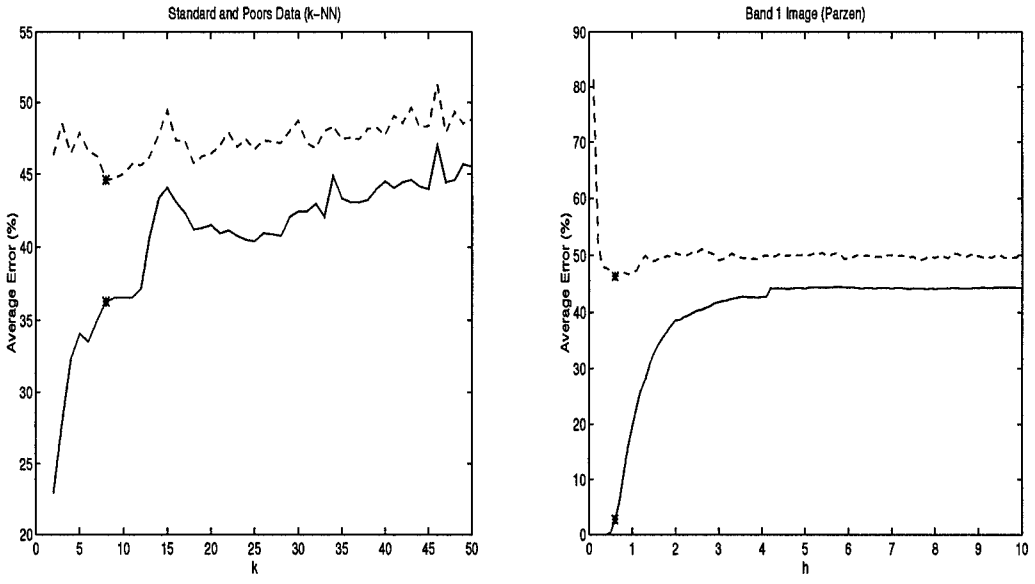


Figure 45. Bayes Bounding of Raw SandP data, m=6

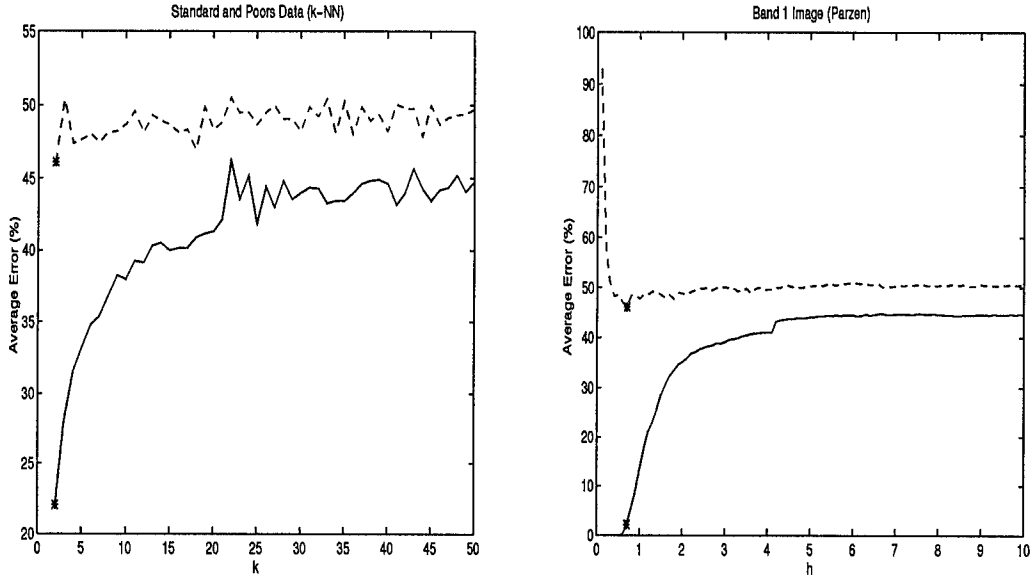


Figure 46. Bayes Bounding of Raw SandP data, m=7

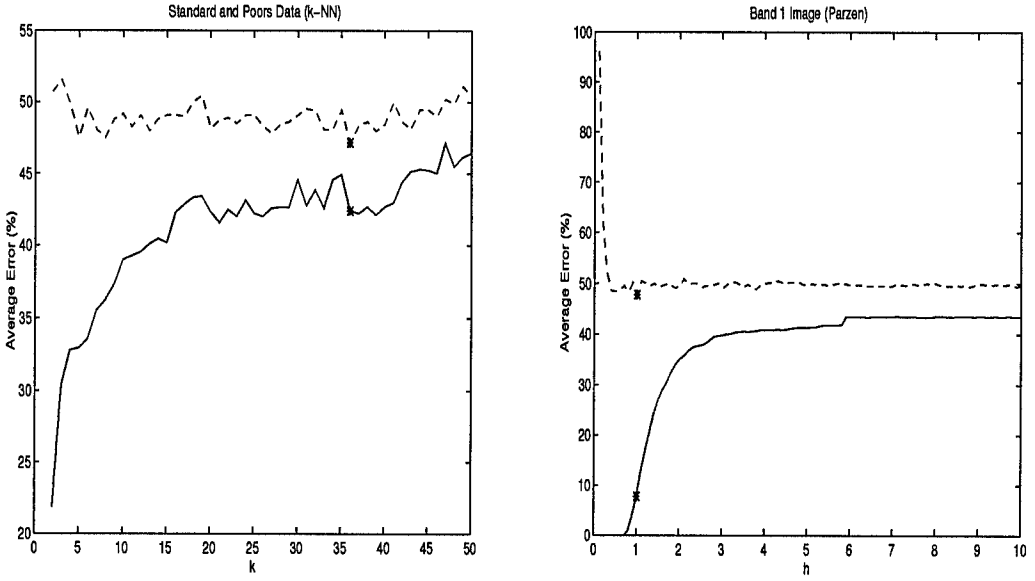


Figure 47. Bayes Bounding of Raw SandP data, m=8

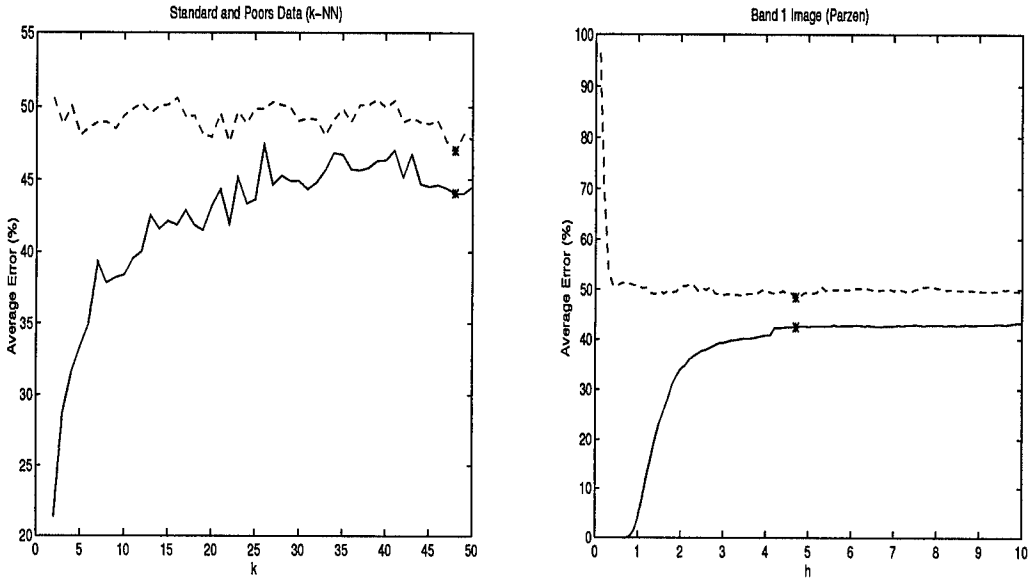


Figure 48. Bayes Bounding of Raw SandP data, m=9

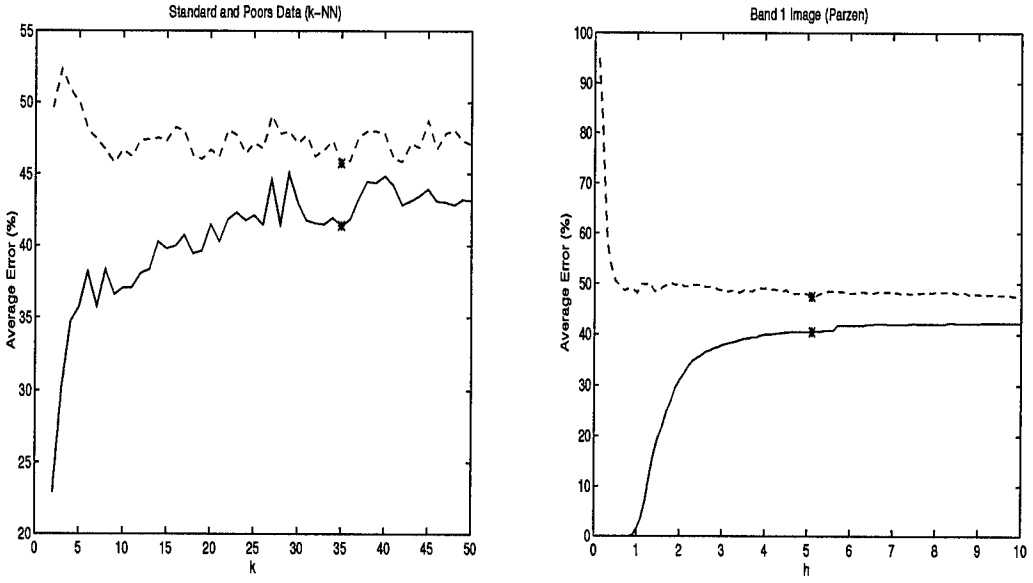


Figure 49. Bayes Bounding of Raw SandP data, m=10

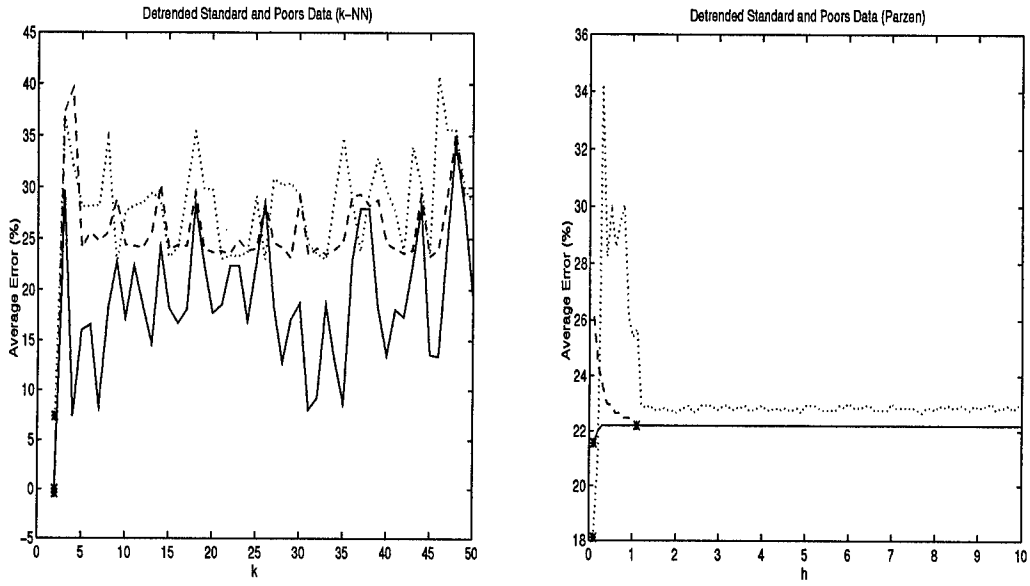


Figure 50. Bayes Bounding of First Differences SandP data, m=1

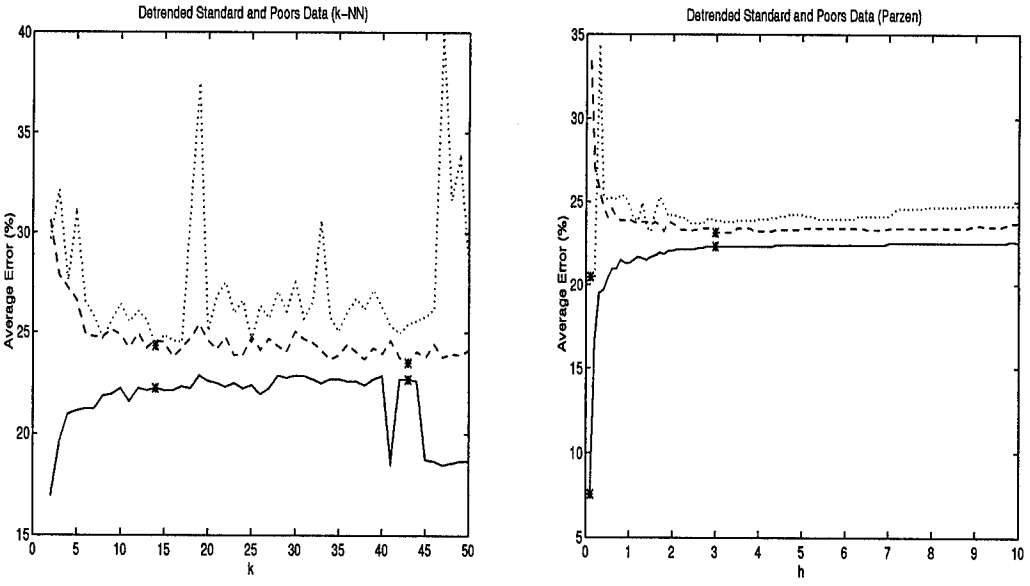


Figure 51. Bayes Bounding of First Differences SandP data, m=2

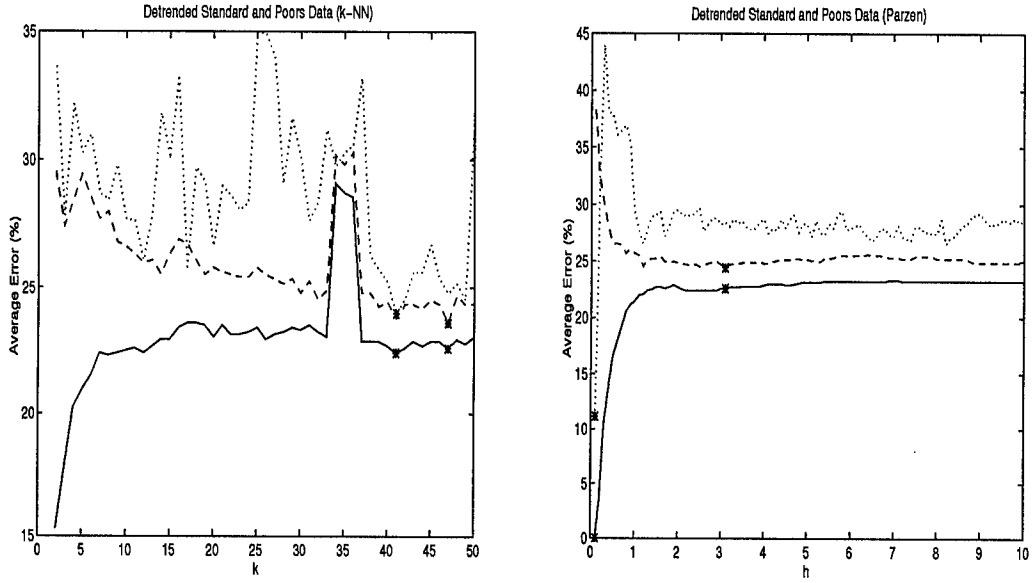


Figure 52. Bayes Bounding of First Differences SandP data, m=3

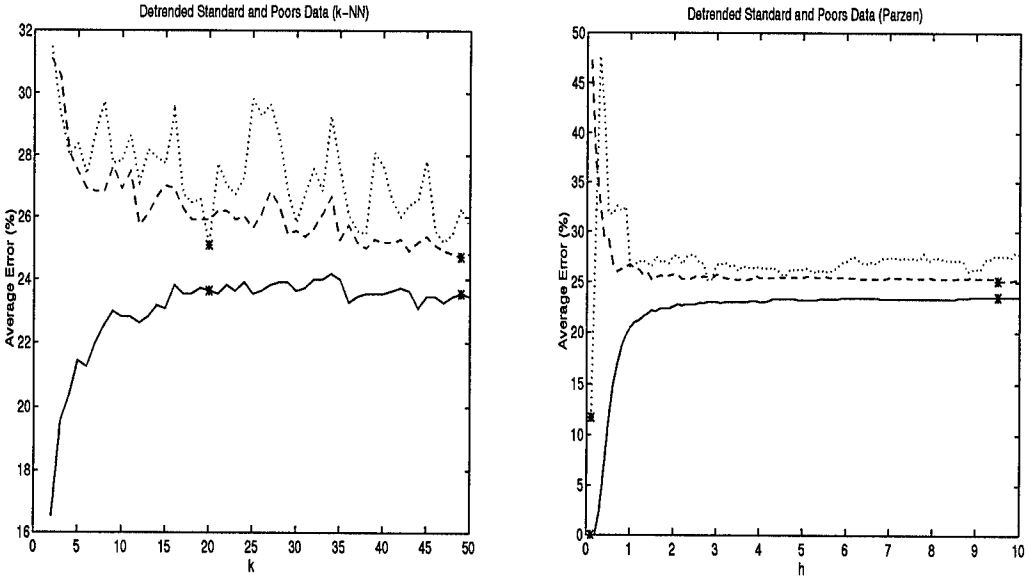


Figure 53. Bayes Bounding of First Differences SandP data, m=4

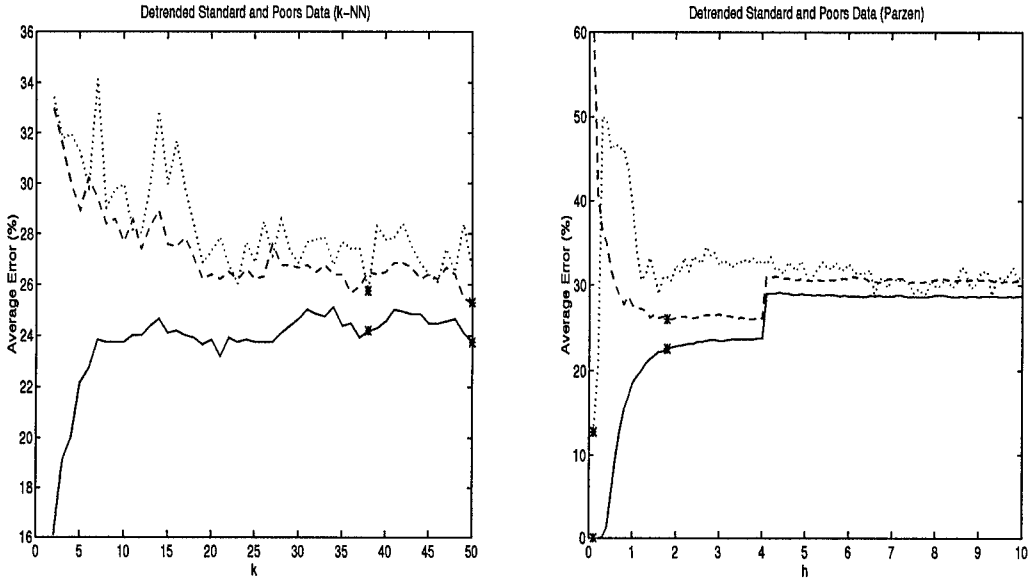


Figure 54. Bayes Bounding of First Differences SandP data, m=5

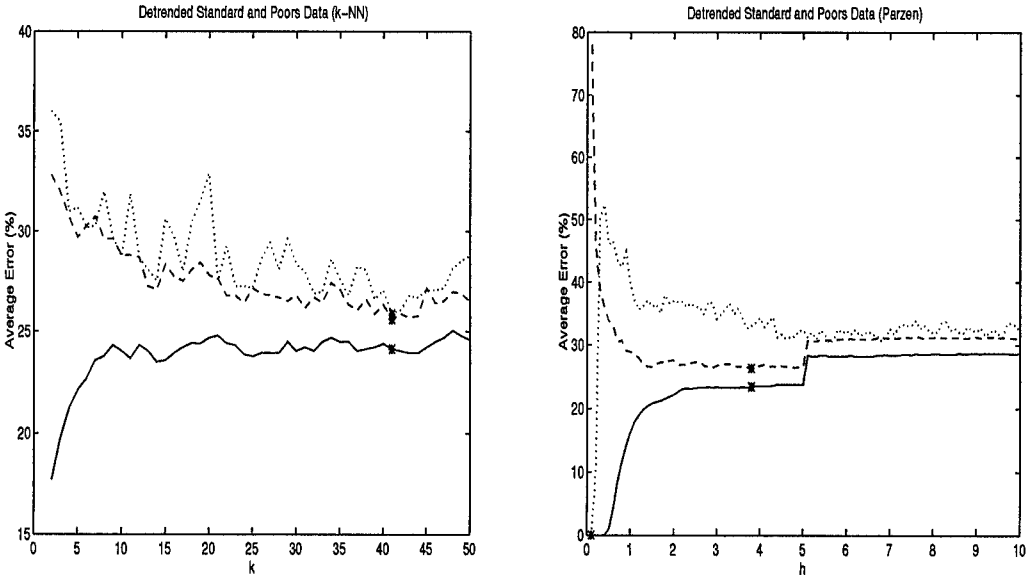


Figure 55. Bayes Bounding of First Differences SandP data, m=6

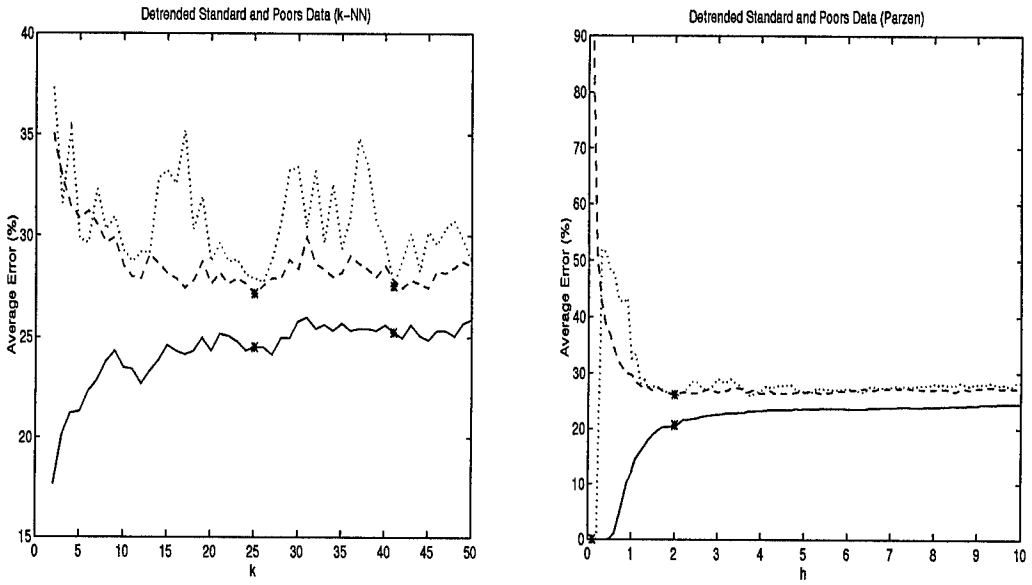


Figure 56. Bayes Bounding of First Differences SandP data, m=7

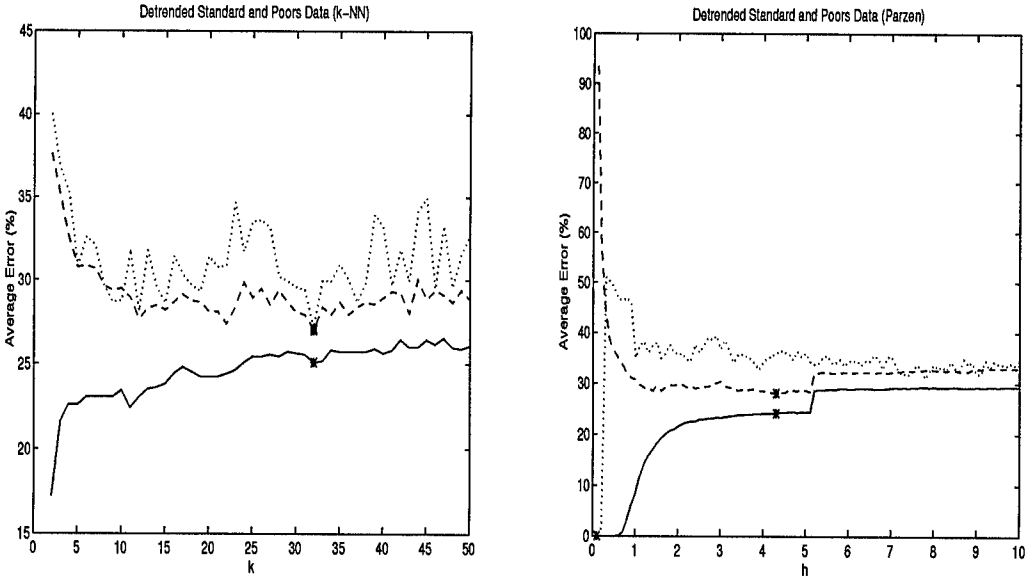


Figure 57. Bayes Bounding of First Differences SandP data, m=8

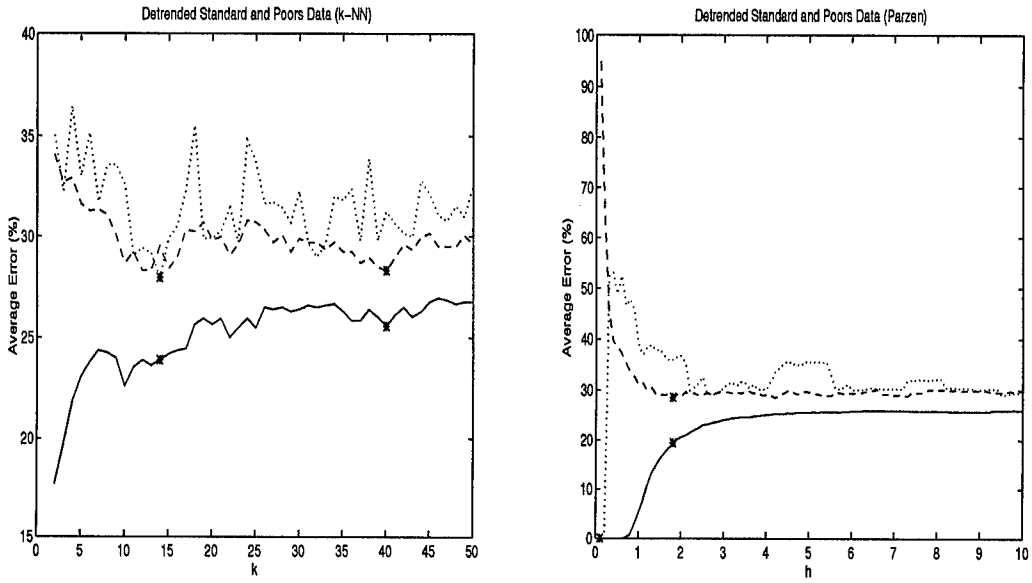


Figure 58. Bayes Bounding of First Differences SandP data, m=9

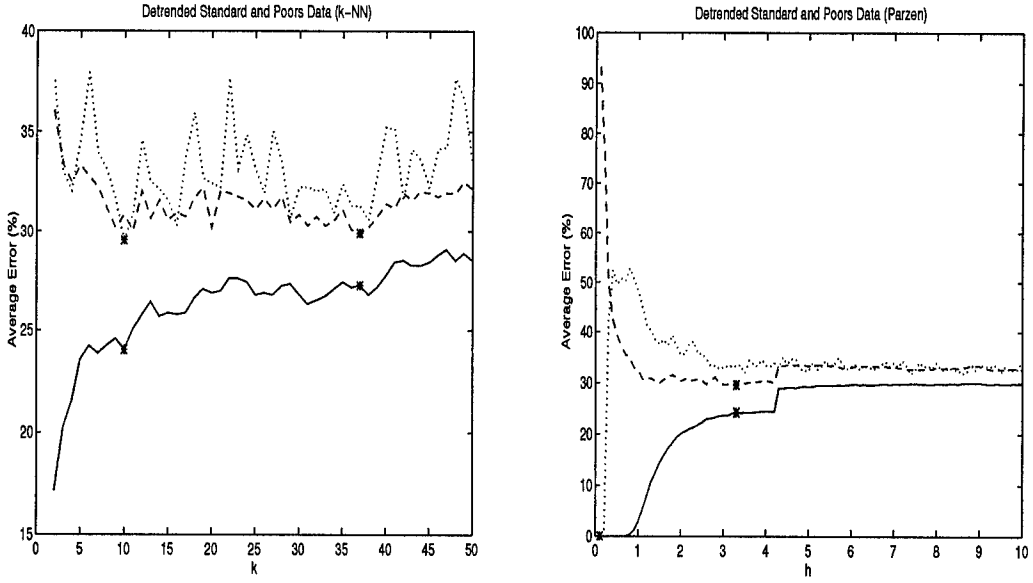


Figure 59. Bayes Bounding of First Differences SandP data, m=10

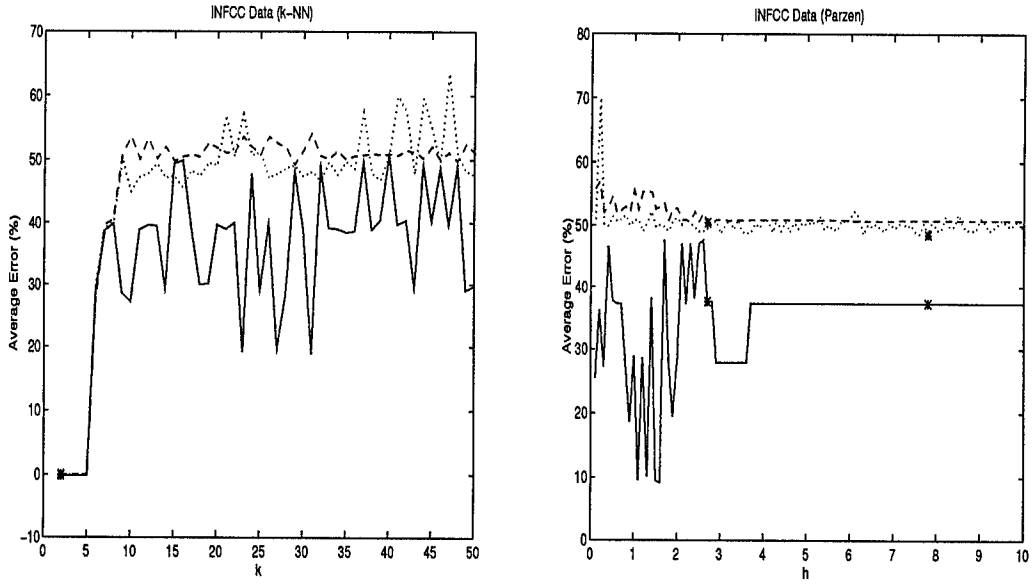


Figure 60. Bayes Bounding of Raw INFFC data, m=1, classes 0,1

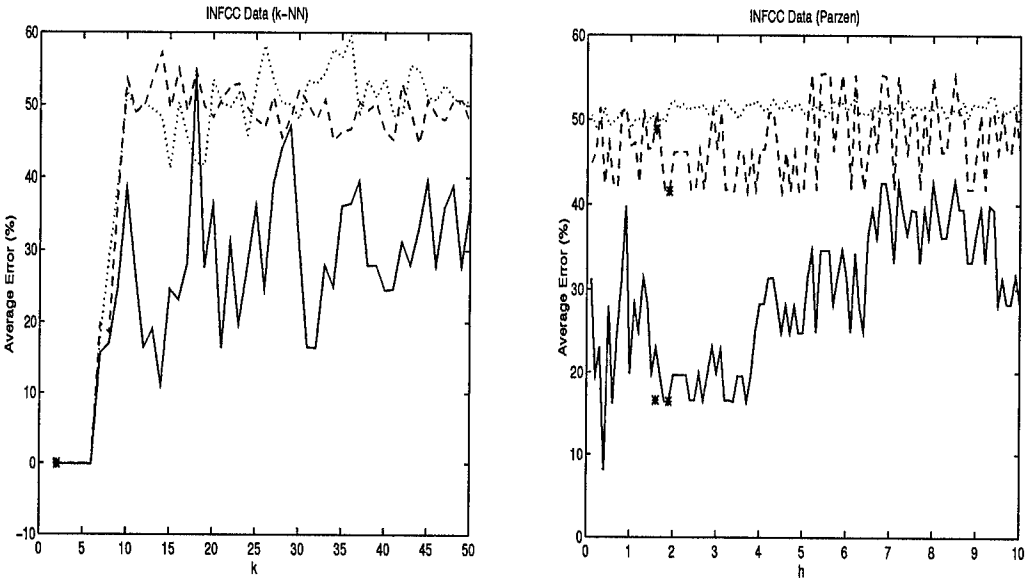


Figure 61. Bayes Bounding of Raw INFFC data, m=1, classes 0,2

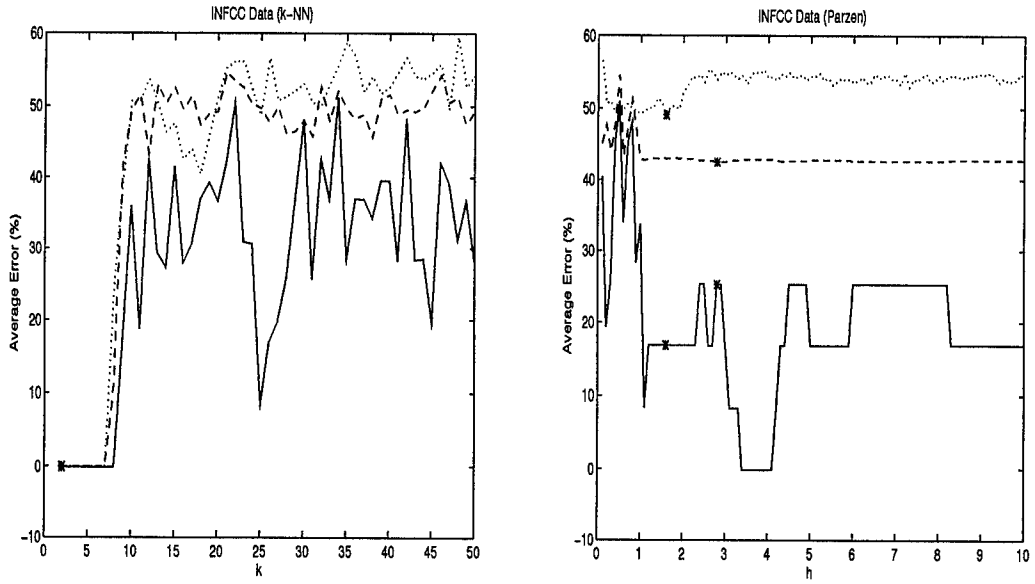


Figure 62. Bayes Bounding of Raw INFFC data, m=1, classes 1,2

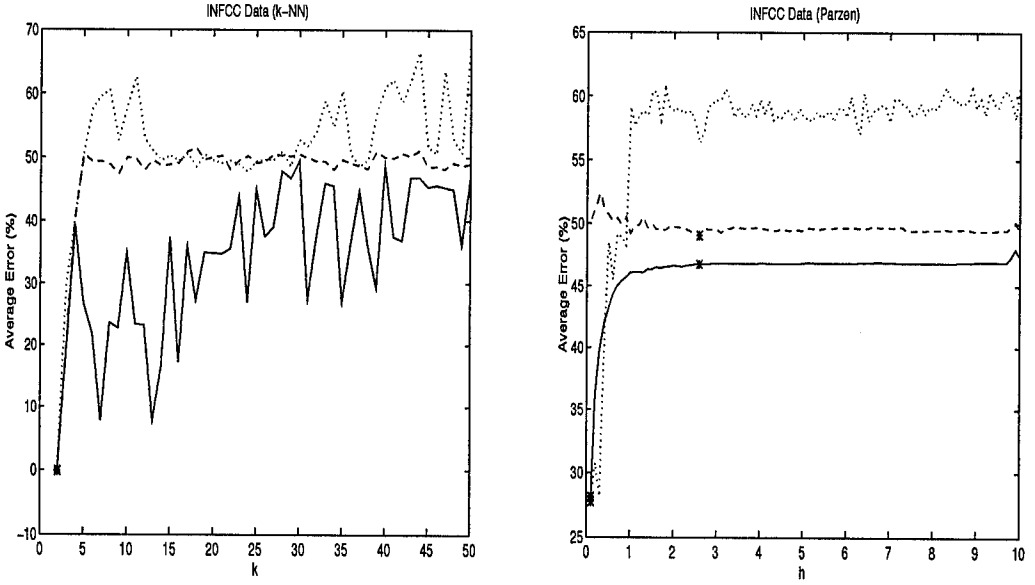


Figure 63. Bayes Bounding of Raw INFFC data, m=2, classes 0,1

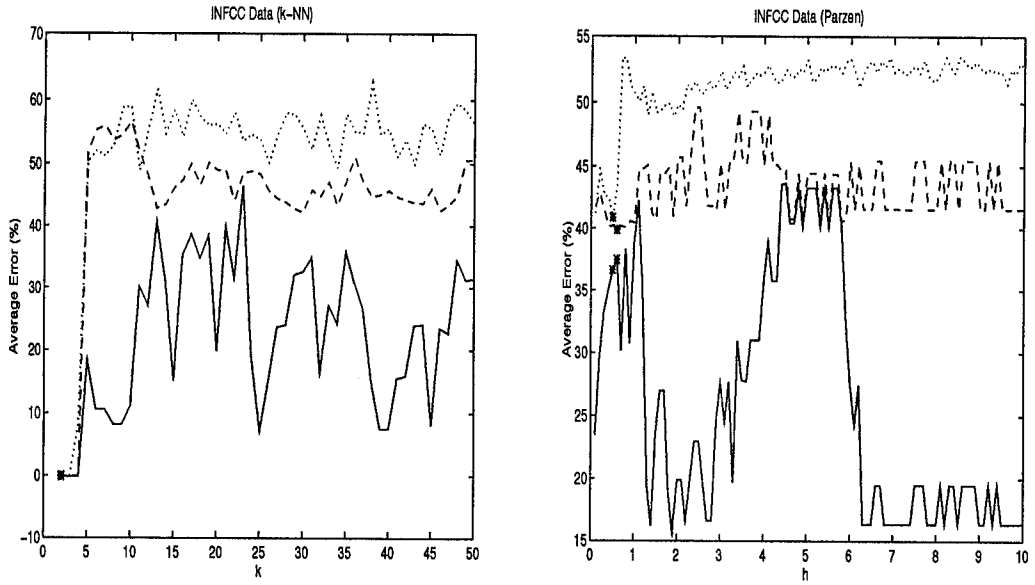


Figure 64. Bayes Bounding of Raw INFFC data, m=2, classes 0,2

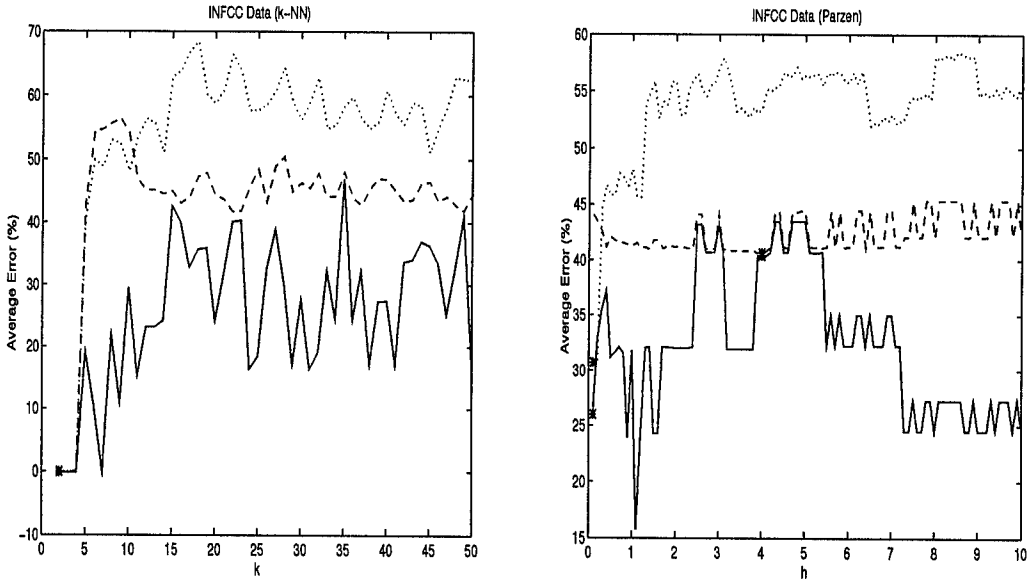


Figure 65. Bayes Bounding of Raw INFFC data, m=2, classes 1,2

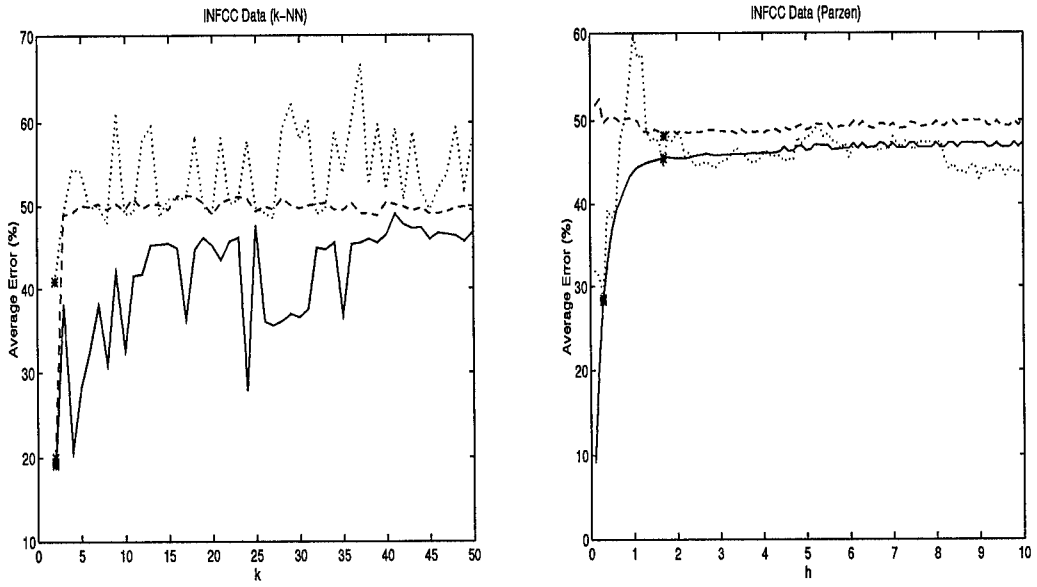


Figure 66. Bayes Bounding of Raw INFFC data, m=3, classes 0,1

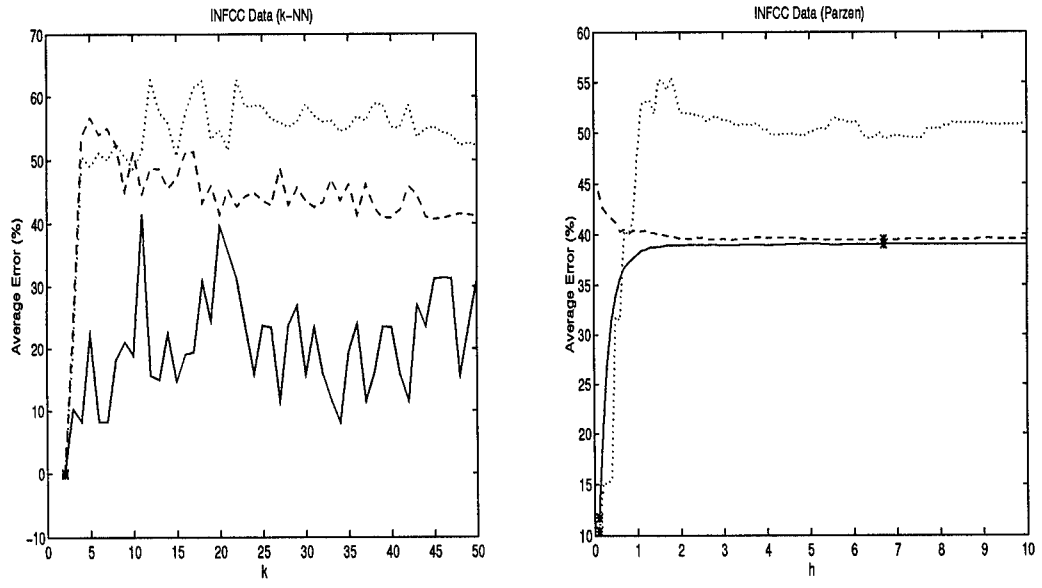


Figure 67. Bayes Bounding of Raw INFFC data, m=3, classes 0,2

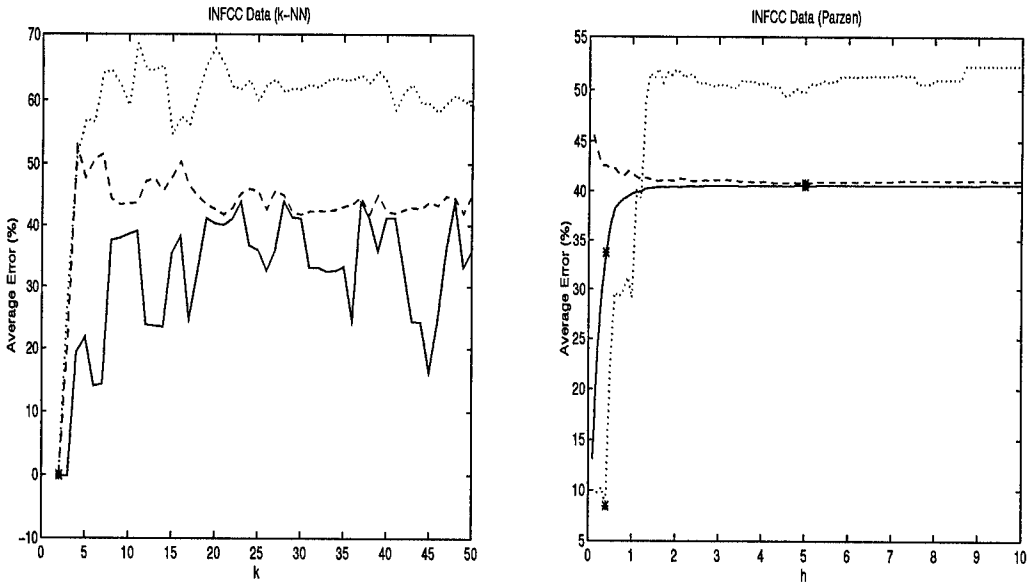


Figure 68. Bayes Bounding of Raw INFFC data, m=3, classes 1,2

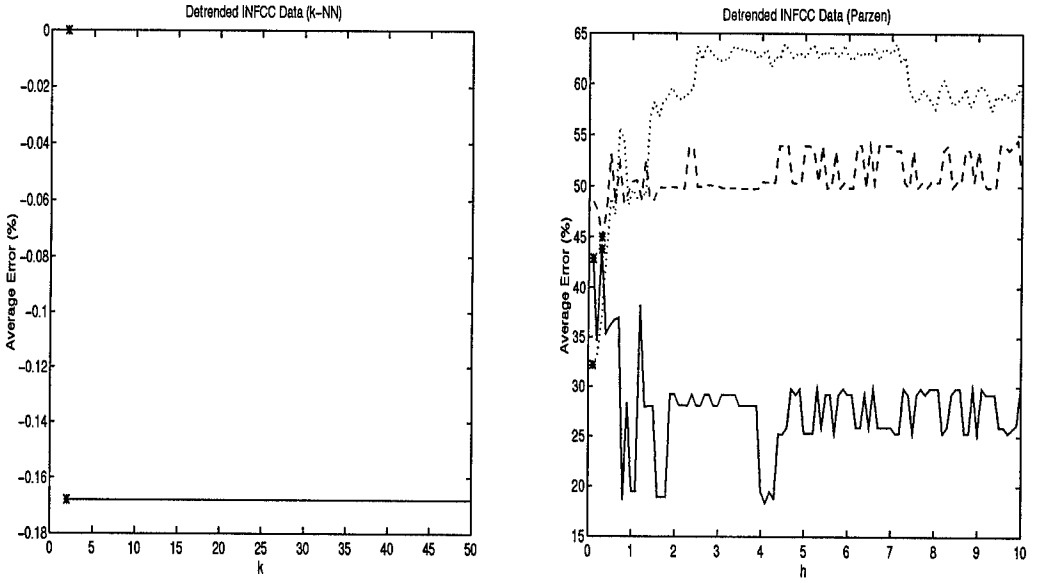


Figure 69. Bayes Bounding of First Differences INFFC data, m=1, classes 0,1

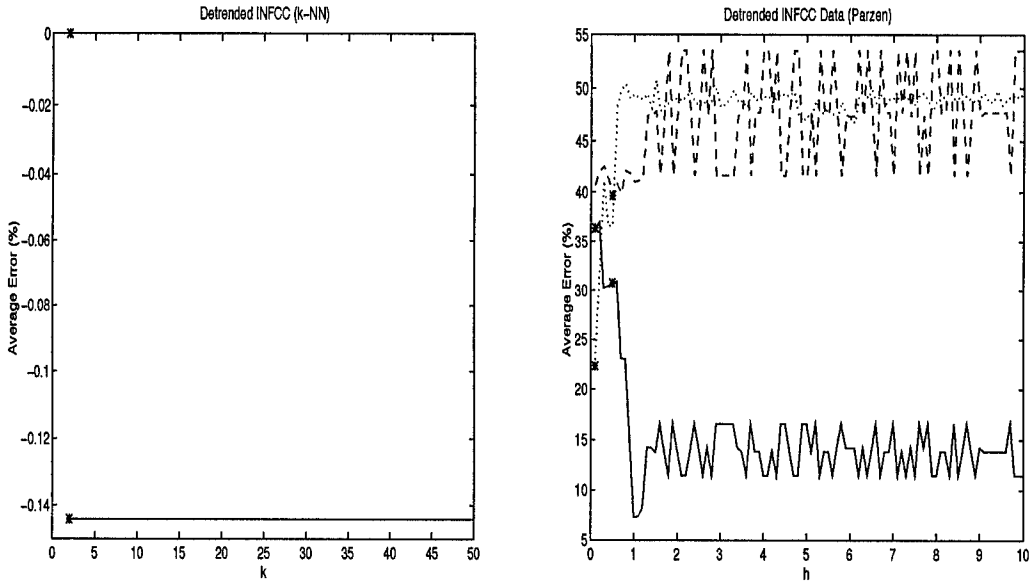


Figure 70. Bayes Bounding of First Differences INFFC data, m=1, classes 0,2

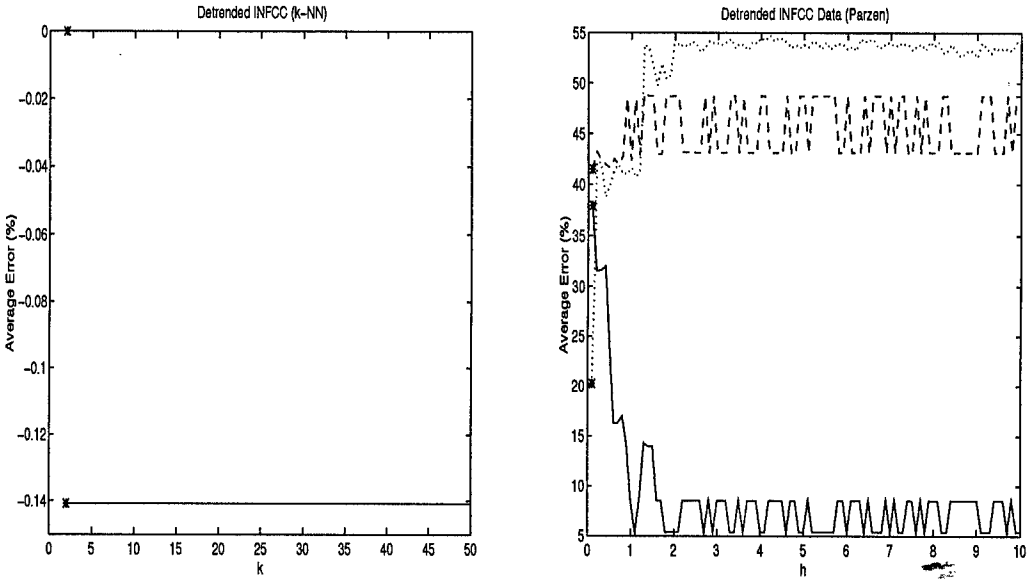


Figure 71. Bayes Bounding of First Differences INFFC data, m=1, classes 1,2

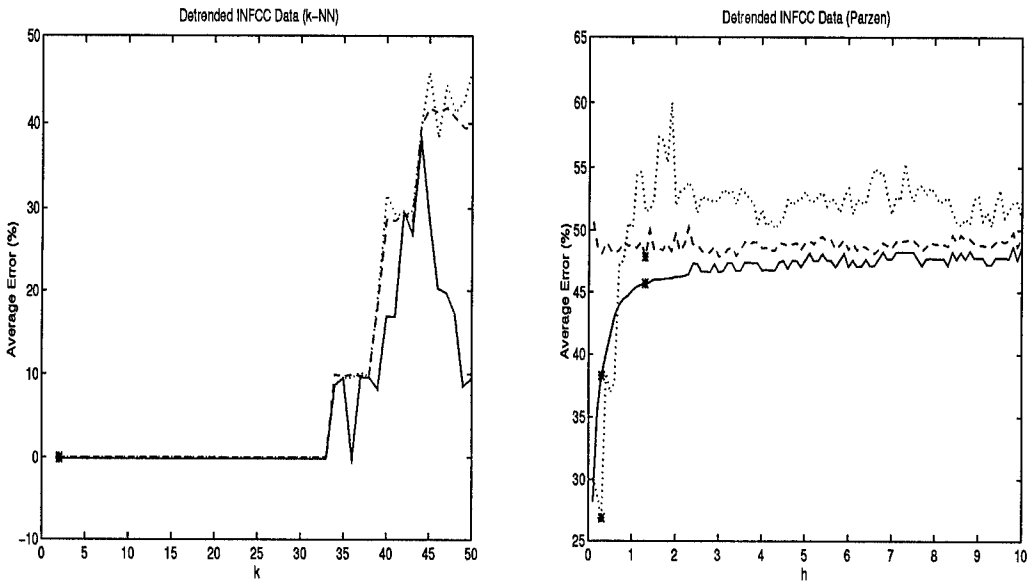


Figure 72. Bayes Bounding of First Differences INFFC data, m=2, classes 0,1

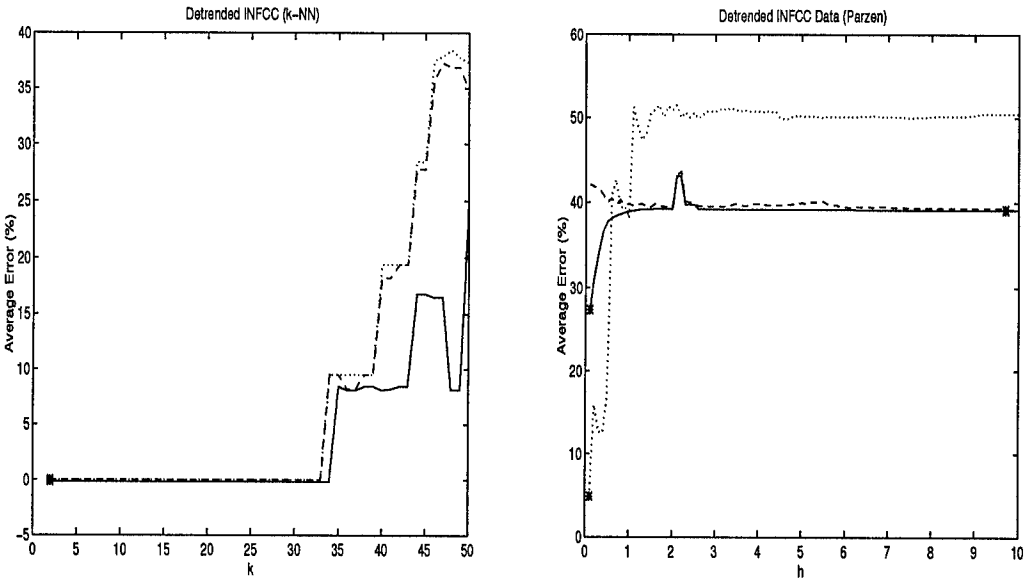


Figure 73. Bayes Bounding of First Differences INFFC data, m=2, classes 0,2

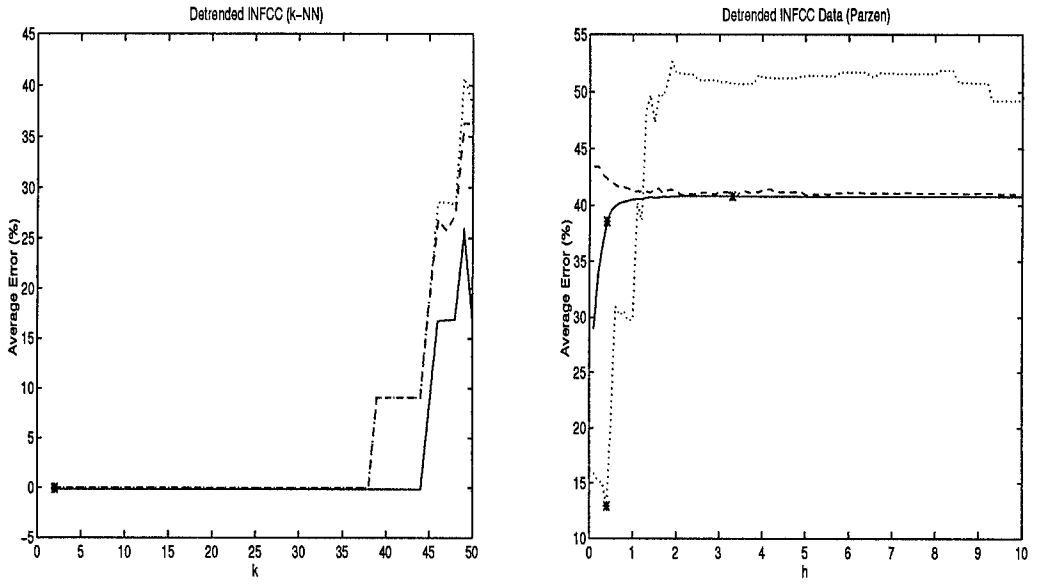


Figure 74. Bayes Bounding of First Differences INFFC data, m=2, classes 1,2

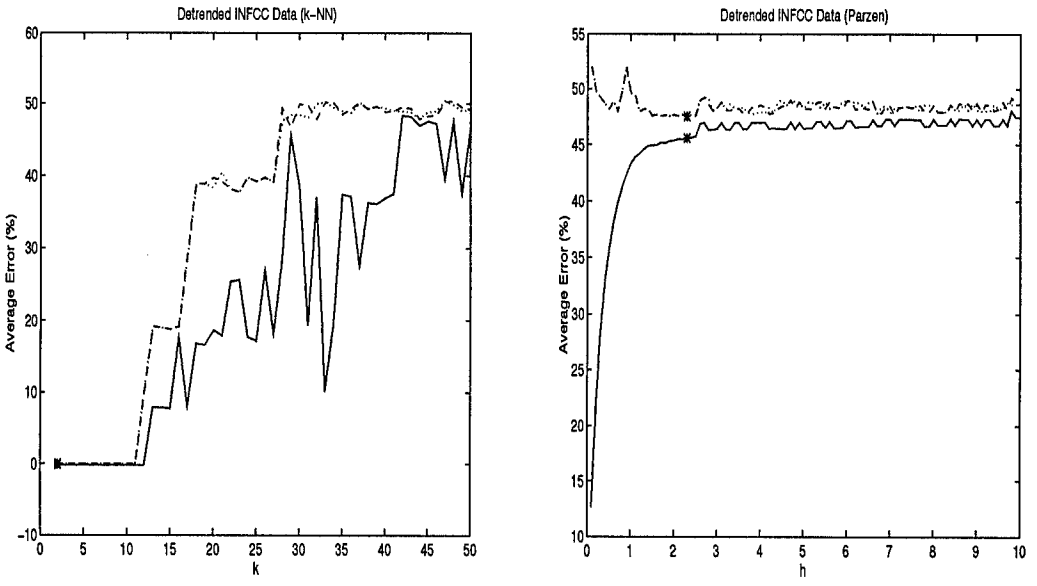


Figure 75. Bayes Bounding of First Differences INFFC data, m=3, classes 0,1

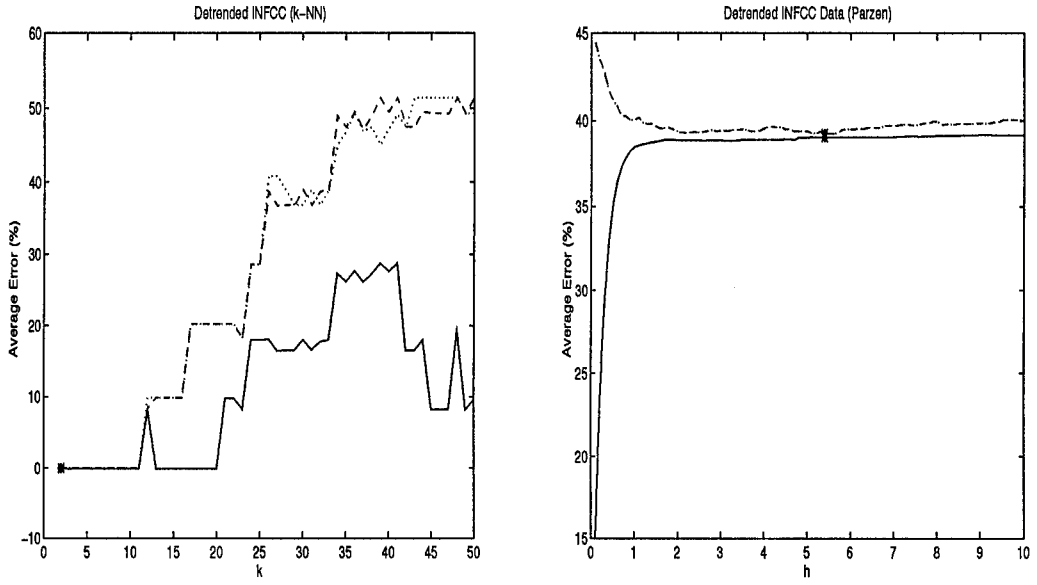


Figure 76. Bayes Bounding of First Differences INFFC data, m=3, classes 0,2

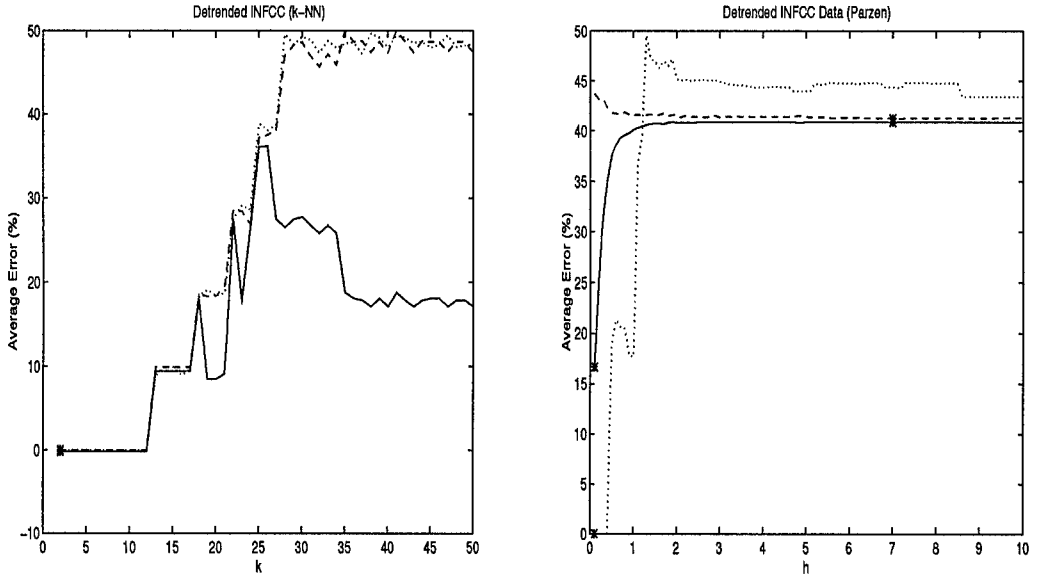


Figure 77. Bayes Bounding of First Differences INFFC data, m=3, classes 1,2

Bibliography

1. Casdagli, Martin C., "Chaos and Deterministic versus Stochastic Non-linear Modelling", *Journal of the Royal Statistical Society B, 54(2)*: 303-328, 1991.
2. Casdagli, Martin C., *et. al.*, "Non-linear Modelling of Chaotic Time Series: Theory and Applications", *Proceedings of Electric Power Research Institute Workshop Applications of Chaos*: 1991.
3. Casdagli, Martin C. and Andreas S. Weigend, "Exploring the Continuum Between Deterministic and Stochastic Modelling", *Time Series Prediction: Forecasting the Future and Understanding the Past* edited by Andreas S. Weigend and Neil A. Gershenfeld: 347-366, Addison-Wesley, 1994.
4. Chan, L.W. and F. Fallside, "An Adaptive Training Algorithm for Back Propagation Networks", *Computer Speech and Learning*: Academic Press, 1987.
5. Colombi, John M., *Cepstral and Auditory Model Features for Speaker Recognition*, MS thesis, Air Force Institute of Technology, 1992.
6. Cybenko, G., "Approximation by Superpositions of a Sigmoidal Function", Research Note, Computer Science Department, Tufts University, October 1988.
7. Farmer, J. Doyne and John J. Sidorowich, "Predicting Chaotic Time Series", *Physical Review Letters 59(8)*, 1987.
8. Fukinaga, Keinosuke and Donald M. Hummels, "Bayes Error Estimation Using Parzen and k-NN Procedures", *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9*, 1987.
9. Gainey, James C. Jr., *Predicting Nonlinear Time Series*, MS thesis, Air Force Institute of Technology, 1993.
10. Garza, Robert E., *Report on Data Manipulation, Plotting, and Prediction*, EE 699 Final Report, Department of Electrical Engineering, Air Force Institute of Technology, 1994.
11. Gleick, James, *Chaos: Making A New Science*, Viking Penguin, New York, 1988.
12. Grassberger, Peter and Itamar Procaccia, "Measuring the Strangeness of Strange Attractors", *Physica D, 9*: 189-208, 1983.
13. Harrup, Georgia K., *ROC Analysis of IR Segmentation Techniques*, MS thesis, Air Force Institute of Technology, 1994.
14. Hush, Don R. and Bill G. Horne, "Progress in Supervised Neural Networks: Whats New Since Lippman", *IEEE Signal Processing*, 1993.
15. Kay, Steven M., *Modern Spectral Estimation: Theory and Application*, Prentice-Hall, New Jersey, 1988.
16. Kukolich, Linda and Richard P. Lippman, *LNKnet Users Guide*, MIT Lincoln Laboratories, July 1993.

17. Lippman, Richard P., "An Introduction to Computing with Neural Networks", *IEEE ASSP Magazine*: 4-22, 1987.

18. Martin, Curtis E., *Non-Parametric Bayes Error Estimation for UHRR Target Identification*, MS thesis, Air Force Institute of Technology, 1993.

19. Packard, N.H., J.P. Crutchfield, J.D. Farmer, and R.L. Shaw, "Geometry from a Time Series", *Physics Review Letters*: 45, 712-16, 1980.

20. Peitgen, Heinz-Otto, Hartmut Jurgens, and Dietmar Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer-Verlag, New York, 1992.

21. Rao, Valluru B. and Hayagriva V. Rao, *C++ Neural Networks and Fuzzy Logic*: 311-329, Management Information Source, New York, 1993.

22. Rogers, Steven K., Matthew Kabrisky, Dennis W. Ruck, and Gregory L. Tarr, *An Introduction to Biological and Artificial Neural Networks*, Air Force Institute of Technology, 1990.

23. Rogers, Steven K.,"Edge Detection for Image Processing", Class Notes, Department of Electrical Engineering, Air Force Institute of Technology, May 1994.

24. Ruelle, D., "Deterministic Chaos: the Science and the Fiction", *Proceedings of the Royal Statistical Society of London A*: 427, 241, 1990.

25. Sacchini, Joseph J., *Development of Two-Dimensional Parametric Radar Signal Modelling and Estimation Techniques with Application to Target Identification*, Ph.D. dissertation, The Ohio State University, 1992.

26. Sauer, Tim, "Embedology", *Technical Report 91-01-008*, Santa Fe Institute, 1991.

27. Sauer, Tim, "Time Series Prediction by Using Delay Coordinate Embedding", *Time Series Prediction: Forecasting the Future and Understanding the Past, edited by Andreas S. Weigend and Neil A. Gershenfeld*: 175-193, Addison-Wesley, 1994.

28. Schalkhoff, Robert, *Pattern Recognition: Statistical, Structural, and Neural Approaches*, John Wiley and Sons, New York, 1992.

29. Scheaffer, Richard L. and James T. MClave, *Statistics for Engineering*, Prindle, Weber & Schmidt, 1982.

30. Stright, James R., *A Neural Network Implimentation of Chaotic Time Series Prediction*, MS thesis, Air Force Institute of Technology, 1988.

31. Stright, James R., *Embedded Chaotic Time Series: Applications in Prediction and Spatio-Temporal Classification*, Ph.D. dissertation, Air Force Institute of Technology, 1994.

32. Takens, Florin, "Detecting Strange Attractors in Fluid Turbulence", *Dynamical Systems and Turbulence*, Warwick 1980, Lecture Notes in Mathematics 898,

edited by D.A. Rand and L.S. Young: 366-381, Springer-Verlag, 1981. Printed in Germany.

33. Tenorio,M.F., *et. al.*, *International Financial Forecasting Competition*, financial data, 1994.

34. Zahirniak, Daniel R., *Characterization of Radar Signals Using Neural Networks*, MS thesis, Air Force Institute of Technology, 1990.

Vita

Lieutenant Robert E. Garza was born on 9 July 1971 in Wolford Hall Medical Center, a little known hospital in San Antonio, Texas. After moving to California, he graduated from Wheatland High School as a co-Validictorian in 1989. He entered the United States Air Force Academy three weeks later. He graduated from the Academy with a Bachelor of Science in Electrical Engineering and a commission in the USAF in 1993. Not ready to face the turmoil of actually having to work for a living, he accepted a selection to attend the School of Engineering, Air Force Institute of Technology.

Permanent address: 6670 Deer Bluff Drive
Huber Heights, OH 45424

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1994	Master's Thesis

4. TITLE AND SUBTITLE
Embedology and Neural Estimation for Time Series Prediction

5. FUNDING NUMBERS

6. AUTHOR(S)
Robert E. Garza

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/ENG/94D-11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Capt. Jim Stright
WL/MNGA
Bldg. 13 101 W. Eglin Blvd. Suite 206
Eglin AFB, FL 32542-6810

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstract

Time series prediction has widespread application, ranging from predicting the stock market to trying to predict future locations of scud missiles. Recent work by Sauer and Casdagli has developed into the embedology theorem, which sets forth the procedures for state space manipulation and reconstruction for time series prediction. This includes embedding the time series into a higher dimensional space in order to form an attractor, a structure defined by the embedded vectors. Embedology is combined with neural technologies in an effort to create a more accurate prediction algorithm. These algorithms consist of embedology, neural networks, Euclidean space nearest neighbors, and spectral estimation techniques in an effort to surpass the prediction accuracy of conventional methods. Local linear training methods are also examined through the use of the nearest neighbors as the training set for a neural network. Fusion methodologies are also examined in an attempt to combine several algorithms in order to increase prediction accuracy. The results of these experiments determine that the neural network algorithms have the best individual prediction accuracies, and both fusion methodologies can determine the best performance. The performance of the nearest neighbor trained neural network validates the applicability of the local linear training set.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Time Series Prediction, Embedology, Neural Networks	177
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

- **DOD** - See DoDD 5230.24, "Distribution Statements on Technical Documents."
- **DOE** - See authorities.
- **NASA** - See Handbook NHB 2200.2.
- **NTIS** - Leave blank.

Block 12b. Distribution Code.

- **DOD** - Leave blank.
- **DOE** - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
- **NASA** - Leave blank.
- **NTIS** - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

★U.S.GPO 1990-0-273-271